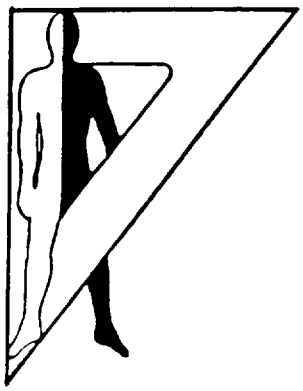

AD

Technical Note 6-92

HEL COUNTER-AIR PROGRAM SIMULATION NETWORKING USING TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL (TCP/IP)

Maria del C. Lopez

June 1992
AMCMS Code 612716.H700011

U.S. ARMY HUMAN ENGINEERING LABORATORY

Aberdeen Proving Ground, Maryland

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Note 6-92	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Human Engineering Laboratory 6b. OFFICE SYMBOL (If applicable) SLCHE	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Aberdeen Proving Ground, MD 21005-5001	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 6.27.16 / PROJECT NO. 1L162716AH70 / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
HEL Counter-Air Program Simulation Networking Using Transmission Control Protocol/Internet Protocol (TCP/IP)

12. PERSONAL AUTHOR(S)
Lopez, Maria del C.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO _____	1992, June	80

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	aviation	C programming	messages
01	02	(see reverse)	command	counter-air	network
01	03	0301	communications	Ethernet	simulation
			control	FORTRAN	TCP/IP

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the C functions and FORTRAN subroutines involved in the communications between the aviation tactical operations center (AVTOC), air defense tactical operations center (ADTOC), and helicopter nodes for the U.S. Army Human Engineering Laboratory (HEL) counter-air program demonstration. It uses transmission control protocol/internet protocol (TCP/IP) as the communications protocol, and it is written under the virtual address extension/virtual memory system (VAX/VMS) and UNIX operating systems in a manned, interactive simulation. The software design consists of ten C programmer written functions and one FORTRAN programmer written subroutine. The functions and subroutine are explained in detail and a listing of source codes is included in the appendices.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified

22. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Technical Reports Office	(301) 278-4478	SLCHE-SS-TSB

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

17. (continued)

12	05
12	06
25	03
25	05

AMCMS Code 612716.H700011

Technical Note 6-92

HEL COUNTER-AIR PROGRAM SIMULATION NETWORKING USING TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL (TCP/IP)

Maria del C. Lopez

June 1992

APPROVED: Charles M. Shoemaker

CHARLES M. SHOEMAKER
Acting Director
Human Engineering Laboratory

U.S. ARMY HUMAN ENGINEERING LABORATORY

Aberdeen Proving Ground, Maryland

CONTENTS

FIGURES

HEL COUNTER-AIR PROGRAM SIMULATION NETWORKING USING TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL (TCP/IP)

BACKGROUND

The U.S. Army Human Engineering Laboratory (HEL) at Aberdeen Proving Ground, Maryland, is the U.S. Army Laboratory Command's lead laboratory for human factors engineering. Increasingly sophisticated equipment and soldier interfaces require HEL to perform effective investigations of complex soldier-machine interfaces.

The HEL counter-air program (HELCAP) was initiated in 1987 with the objective of optimizing soldier-machine interfaces in command control networks that integrate Army air and counter-air operations. HELCAP provides a warfighter-in-the-loop simulation and provides a focus on the command, control, and communications (C^3) issues in a limited combination of aviation and air defense teams. This simulation configuration includes four manned nodes: a helicopter node run on the VAX 6410, an air defense tactical operations center (ADTOC) node run on the Silicon Graphics IRIS 3130, an aviation tactical operations center (AVTOC) node run on another Silicon Graphics IRIS 4D/85GT, and an integrated weapons system display/pedestal-mounted stinger (IWSD/PMS) node run on the Silicon Graphics IRIS 4D/85GT. Each of these nodes is described briefly in the following paragraphs.

The helicopter node in the HELCAP demo uses the cockpit research experimentation and work load simulator (CREWS). This is a low cost, warfighter-in-the-loop, real-time helicopter simulator for research into the human factors issues that affect the development of new helicopters. CREWS provides a method to evaluate the human factors issues in a part task simulation within a crew station environment. (Note. Part task simulation is a term used when the simulation does not provide all the tasks involved in the actual system.)

CREWS is designed as a flexible, generic, fixed base, single place crew station capable of being quickly reconfigured. Four small cathode ray tube (CRT) monitors and a helmet-mounted display (HMD) are used to present vertical and horizontal situation, subsystems, mission management, electronic maps, and air situation displays. Additionally, a variety of control input devices is available, which can be selected to comprise a special cockpit research configuration. These devices include special key pads, function keys, joysticks, speech recognition, and speech output. An external scene generator, an advanced version of the low cost systems used in the simulator net (SIMNET) simulators, provides the pilot subjects with a 40° by 40° out-the-window scene. Currently, the out-the-window scene models an 11-km by 18-km region of the Fulda Gap in Germany. Texture capabilities provide a highly realistic scene, which consists of trees, roads, buildings, rivers, and hills. Static and dynamic objects such as other helicopters and fixed wing aircraft are included as well as military tanks, trucks, and other ground vehicles.

The host computers for the CREWS consist of a MicroVAX II, PDP 11/23 with array processor and a VAX 6410. These computers are networked by the Ethernet local area computer network for computer process control and data exchange.

The ADTOC and AVTOC nodes are identical in design. These TOCs are real-time, warfighter-in-the-loop simulators. The designs are not human factored but are provided as vehicles to explore the counter-air operations and

communications procedures that may define future TOC designs for counter-air operations.

The operator stations for the ADTOC and AVTOC consist of a 19-inch diagonal high resolution color monitor, an alphanumeric keyboard, and a conventional mouse. The screen display consists of three functional areas: a battlefield situation display (BSD), which occupies the major portion of the screen; a message region at the right side of the screen; and an interactive area at the bottom of the display, which enables the operator to perform selected operations on other areas of the display. Scaling and zooming of the BSD are provided, as well as declutter capabilities.

The ADTOC node is implemented on a Silicon Graphics model IRIS 3130 work station (SGI IRIS 3130). A 19-inch color CRT monitor with a resolution of 1024 pixels x 768 pixels and a refresh rate of 60 Hz is used to display the tactical situation.

A Silicon Graphics model IRIS 4D/85GTB is used to implement the AVTOC node. A 19-inch color monitor with a resolution of 1280 pixels x 1024 pixels and a refresh rate of 60 Hz is used to display the tactical situation.

The remaining node in the HELCAP simulation is a pedestal-mounted stinger (PMS) simulator, which is called the IWSD/PMS node. It provides a real-time, warfighter-in-the-loop capability to simulate an air defense fire unit and is manned by a PMS gunner. The IWSD/PMS node will also demonstrate an IWSD concept.

A Silicon Graphics model IRIS 4D/85GT with alpha overlay planes is used to implement the IWSD/PMS node. A 9-inch color monitor with a resolution of 640 pixels x 480 pixels and a refresh rate of 30 Hz is used for the IWSD/PMS operator.

The transmission control protocol/Internet protocol (TCP/IP) is used to communicate between nodes using the Ethernet local area network. HELCAP communication between nodes is primarily digital and uses the format defined by the forward area air defense (FAAD) data link technical interface design plan (MICOM, 1988). Simulated voice radio can also be used. On the air defense side, communications occur between the ADTOC and other air defense units and the AVTOC. On the aviation side, the AVTOC communicates with other aviation units and the ADTOC. Communication paths among units in counter-air operations in the HELCAP design are limited. For example, there are no direct communication links between a helicopter flight and a IWSD/PMS squad. Communications affecting a flight or an air defense squad first go to their associated TOC, then pass between TOCs.

A 38-minute scenario has been developed for the HELCAP gaming area. A computer data base describes the tactical situation for each 2-second step of the scenario. Each node executes its own local copy of the scenario data base using a scenario generator. The local copy of the data base is adjusted as targets are destroyed, by means of messages sent from the destroying node to all other nodes.

As the scenario runs, it provides airborne track updates at 2-second intervals for 103 aircraft operations, both friendly and hostile, and for static hostile and friendly ground force situations. The overall gaming area for the HELCAP simulation is a 180-km x 160-km region of the Fulda Gap area of Germany and is enclosed by map coordinates MB333995 on the northwest, PB133995 on the northeast, MA333395 on the southwest, and PA133395 on the southeast.

Each of the four HELCAP nodes may include all or some part of this gaming area.

The gaming areas for the ADTOC and AVTOC are 180 km x 160 km, and each covers the same area as the overall HELCAP gaming area. The ADTOC is located at map coordinate NB333295, and the AVTOC is located at map coordinate NB065335. The IWSD/PMS node is located at map coordinate NB165212 at an altitude of 325 meters above mean sea level. Its gaming area extends outward for a distance of 30 km in all directions from its fixed location.

The visual range for the IWSD/PMS infrared system and unaided eye extends outward in all directions for a distance of 9 km from the IWSD/PMS location.

Unlike the other nodes, which are at fixed positions during the simulation the helicopter node, CREWS is free to move in a data base region developed for the external visual scene that is 11 km x 18 km. This external scene gaming area is within the simulation gaming area and is bound by map coordinates NB240320 on the northwest, NB350320 on the northeast, NB240140 on the southwest, and NB350140 on the southeast.

Although CREWS is limited to flight operations within the external scene gaming area of 11 km by 18 km, its simulated sensor system extends the total gaming area outward from the helicopter location for a distance of 40 km in all directions. The cockpit displays include an air-picture display for the helicopter pilot, which displays all air tracks reported by the simulated HELCAP sensors within 40 km in all directions from the helicopters position within the external scene gaming area.

A common reference point for the simulation, the data link reference point (DLRP), has been selected to be at 5,600 kilometers north of the equator and 500 kilometers east of the Greenwich prime meridian at map coordinate NB000000. This provides a common reference for all HELCAP sensor simulations. The DLRP is a point at mean sea level which is the center of a plane containing a master grid oriented on True North. All HELCAP simulated sensors report target positions in X, Y, and elevation relative to the master grid coordinate.

INTRODUCTION

In 1987, HELCAP was initiated with the objective of optimizing soldier-machine interfaces in command control networks that integrate Army air and counter-air operations. The resulting simulation, demonstrated in July 1991, focused on the command, control, and communications (C^3) issues including soldier-in-the-loop. This simulation provided four nodes. Designs for these nodes provided a vehicle to explore the counter-air operations and communication procedures that will define the requirements for future designs.

This report describes the C functions and FORTRAN subroutines involved in the communications between the AVTOC, ADTOC, and helicopter nodes for the HELCAP demonstration. (C programs are comprised of user-defined and run-time library functions.) Communications between the ADTOC and the PMS are discussed by Herald (in press, 1992). Graphics software for the AVTOC and ADTOC running on the Silicon Graphics is discussed by Ware (in press, 1992).

The objective of this report is to provide internal documentation and also to provide a description of the design for others desiring to implement

this or similar software using TCP/IP protocol and written in C and FORTRAN languages under the virtual address extension/virtual memory system (VAX/VMS) and/or UNIX operating systems in a manned, interactive simulation.

The source code for the functions, the include files, and the linker files for the communications software are included in the appendices.

SOFTWARE CONFIGURATION

Information such as enhanced position location reporting system (EPLRS) messages between AVTOC and the helicopter is transmitted during the HELCAP simulation through the interaction of two processes, **commologic** and **commo2**, on the VAX 6410 and function **check_environment** in process **bsd** on the Silicon Graphics 4D/85GT every 2 seconds as shown in Figure 1. (A process is the execution of a program image.)

The **commologic** process employs graphics software to display to the pilot information related to incoming and outgoing messages. The **commo2** process picks up any outgoing messages generated by the **commologic** process and transfers them to the AVTOC. It also picks up incoming messages from the AVTOC and transfers the information to the **commologic** process to be displayed to the pilot. Similar information is transferred between the ADTOC and AVTOC through the interaction of the **bsd** process running on the Silicon Graphics IRIS 4D/85GT and the **bsd** process running on the Silicon Graphics IRIS 3130.

These applications use the TCP/IP as the communications protocol. TCP/IP was preferred to DECnet because no DECnet implementation is available for the Silicon Graphics IRIS 3130. (DECnet is a collective name for the family of communication products [software and hardware] that allow DIGITAL operating systems to participate in a network.) The Silicon Graphics IRIS 4D/85GT and the VAX both run TCP/IP and DECnet (standard protocol on VMS computers) communications protocols. TCP/IP runs as the standard communications protocol on the Silicon Graphics computers.

The TCP/IP software uses sockets to establish or create network-addressable logical entities to define communication points on interconnected machines. For the HELCAP demonstration, a socket was created to interconnect the AVTOC Silicon Graphics IRIS 4D/85GT with the VAX 6410; another socket was created to interconnect the ADTOC Silicon Graphics IRIS 3130 with the IWSD/PMS Silicon Graphics IRIS 4D/85GT.

The **commo2** process runs under the VAX 6410 VMS Version 5.3 operating system. The **bsd** process runs under the Silicon Graphics IRIS 4D/85GT UNIX version 3.1 and IRIS 3130 version 3.6 operating systems. The TCP/IP software running on the VAX is version 5.1. For more information about TCP/IP, refer to Wollongong (1989) and Silicon Graphics (1990).

Memory Management and Differences Between Silicon Graphics and VAX Computers Applied to the Functions

Both processes **commo2** (on the VAX) and **bsd** (on either Silicon Graphics) store the messages in a character array of 12 bytes which share (union) the memory location with different structures that describe each message field. See Appendix A, field.h, for structure declaration. For each message structure declaration, there are fields called "spare#." Some spares

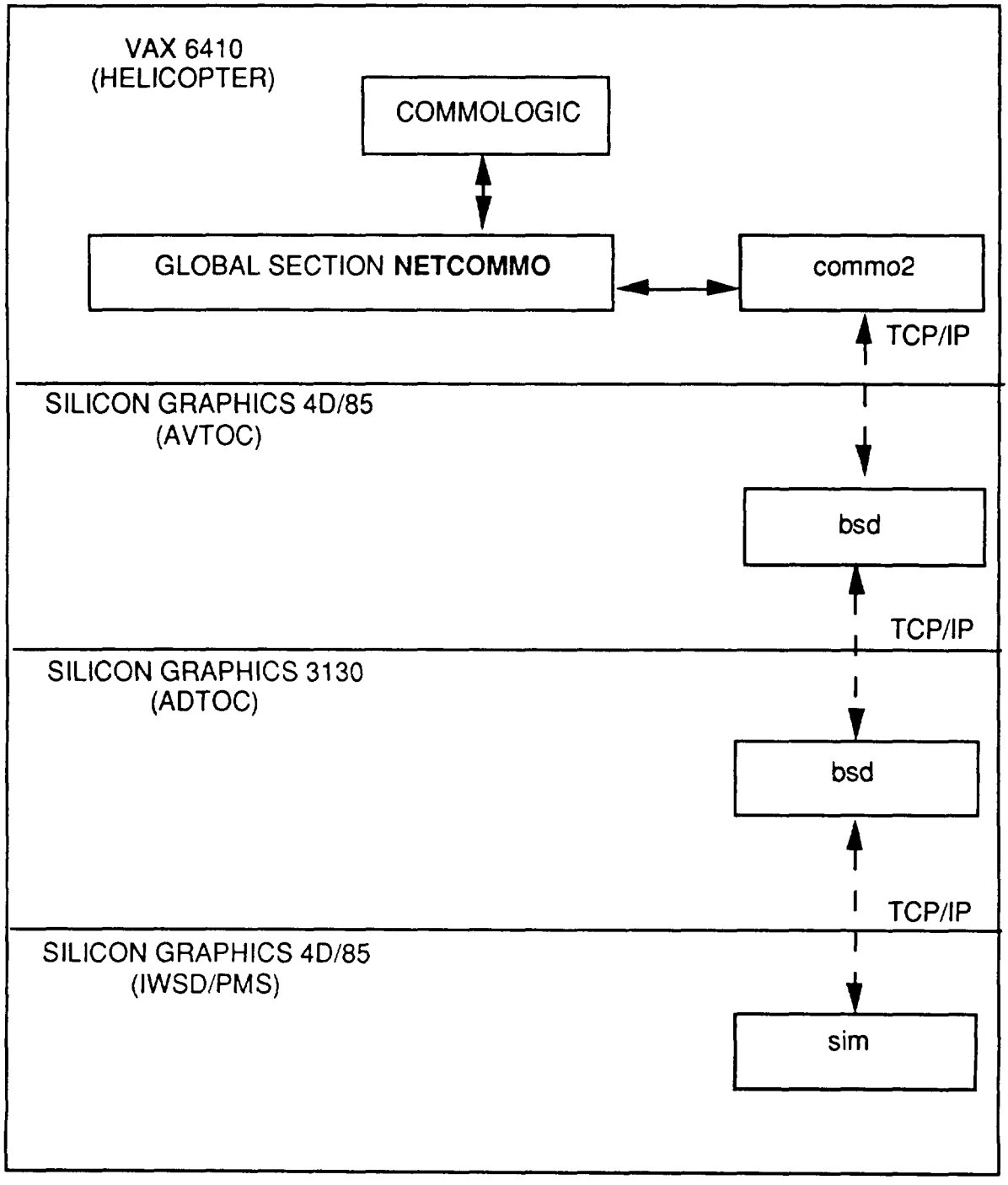

Figure 1. HELCAP network configuration.

are implemented as part of the message, but others are used as padding for the bit fields. Normally, the bits are aligned on word boundaries on the Silicon Graphics IRIS 4D/85GT (32-bit word) and IRIS 3130 (16-bit word). Padding is done automatically by the Silicon Graphics. When one of the bit fields does not fit at the end of the word, the rest of the bits in that word are padded and the field is started on the next word.

The VAX, as opposed to the Silicon Graphics, packs the sequences of bit fields as tightly as possible. Padding is only done on unnamed fields and is aligned on byte boundaries.

Because of the differences in bit field alignment among these three computers, spares are used to fill the possible gaps between fields and to complete 12 bytes whenever the message fields occupy fewer than 12 bytes.

Another important fact is the memory management on the VAX and the Silicon Graphics. The VAX aligns the bytes from right to left starting with byte 0 on the right, as opposed to the Silicon Graphics that aligns the bytes from left to right starting with byte 0 on the left. Because of this difference in memory management, the bytes are shifted on the VAX before the information is sent and after it has been received to or from the Silicon Graphics.

Message Structure Declaration

The transmitted messages have a variable number of bit fields. A union of structures is used to declare each message. For example, the Kill Report message has 6 bits for sublabel code, 1 bit for machine receipt code, 5 for the number of fixed wing killed, 5 for the number of rotary wing killed, 5 for the number of missiles killed, 10 for the rounds expended, 4 for the missiles expended, and 44 as spares--80 bits in all. An example of the Kill Report message as it appeared on the display is shown in Figure 2.

The structure in the VAX program looks like this:

```
struc f16 {
   unsigned spare4:16;
   unsigned spare1:16;
   unsigned spare2:20;
   unsigned sourceid:8;
   unsigned missexp:4;
   unsigned roundexp:10;
   unsigned misskill:5;
   unsigned rotkill:5;
   unsigned fixkill:5;
   unsigned machine:1;
   unsigned sublabel:6;
}buffer16;
```

For purposes of the HELCAP demonstration and to adjust the structure size to a byte boundary the "sourceid" field was added and the spares were extended to make the message 96 bits long so that it fits in the character array of 12 bytes to be transferred between machines.

All messages share the sublabel field, which is declared in the same manner. The sublabel field is used in the functions to identify the message type being transferred.

The structure declaration for the messages on the Silicon Graphics computers is similar to the VAX structure except that the members are in reverse order from top to bottom, because of differences in memory management.

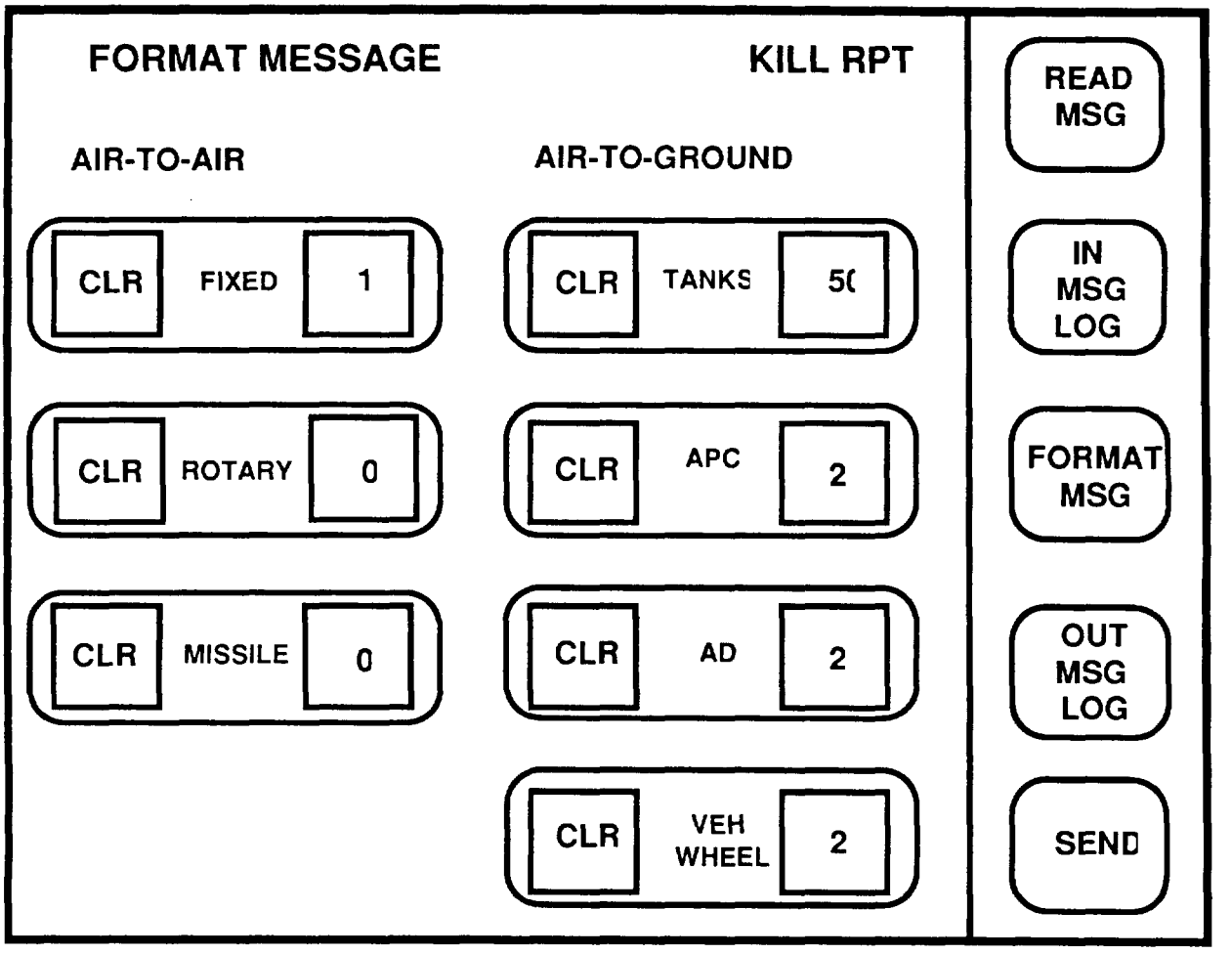

Figure 2. Example of a message (kill report) as presented to the pilot.

INCLUDE FILES

The include files contain information that is used in more than one function. These files are created separate and included (with an include statement) in the function that requires it saving the programmer the job of typing and debugging this information several times during the software development. Six of the eight include functions used in this software are already implemented in the system such as **errno.h**, **in.h**, **netdb.h**, **socket.h**, **time.h**, and **types.h**. The other two, **commo.h** and **field.h**, are programmer written functions used to declare information shared between functions.

The code for these include files appears in Appendix A.

FUNCTION COMMO2

The main function **commo2** runs as a process on the VAX and is called as a function on both Silicon Graphics computers. The following explanation of the **commo2** function is from the point of view of the VAX. The only difference between this function and the one running on the Silicon Graphics is the call to the VAX/VMS system services such as **sys$ascefc, sys$clref, sys$setef, sys$mgblsc**, and **sys$crmpsc**. For more information about system services, refer to DEC (1989).

Function **commo2** associates an event flag cluster number 3 which includes flags 96 through 127 on the VAX. These flags are shared with the FORTRAN processes **commologic, startflights, message**, and **commo2** also on the VAX. It initializes the values of status, start_indicator, aanetcommo.sdrop_track, aanetcommo.stn, timeout.tv_sec, timeout.tv_usec, head_pointer, tail_pointer, and buffer00.buffer.

After the initialization, the process gets into a loop that ends when the variable start_indicator is set to one; in this loop, the function **check_environment** is executed. In this function, the variables of the FORTRAN common labeled "aanetcommo" share the same memory location as the members of the C language structure with variable name aanetcommo. This is why the variables shared with the FORTRAN processes are addressed with the name "aanetcommo." as prefix in the C program **commo2**.

Notice that some lines of code have been commented in function **check_environment**. The value of start_simulation in function check_environment was changed by an unknown source during the simulation test. Lack of time to solve this problem caused the commenting of some lines where the value of start_simulation was critical. Later tests will consider this problem in the early stages of the software development.

The **commo2** process is the main function involved in the communications between ADTOC, AVTOC and helicopter nodes. It consists of ten C programmer written functions: **assign_keyboard, create_sock, closedown, establish_connection, connect_to_host, insert, wait_for_connection, retrieve, check_usage**, and **check_environment** and one FORTRAN subroutine, **create_global**. These functions are described in the following pages and diagrammed in Figure 3, and the source code is given in Appendix B.

The flowchart for the function **commo2** is shown in Figure 4.

FUNCTION ASSIGN_KEYBOARD

This function makes the keyboard serve as an input device for selection of messages and is called in the function **commo2**; it was used for testing purposes in the development process. This function is not called as part of the HELCAP demonstration.

The following variable is used:

keyboard - Integer variable and channel assigned to the keyboard device and opened as input device.

The flowchart for the function **assign keyboard** is shown in Figure 5.

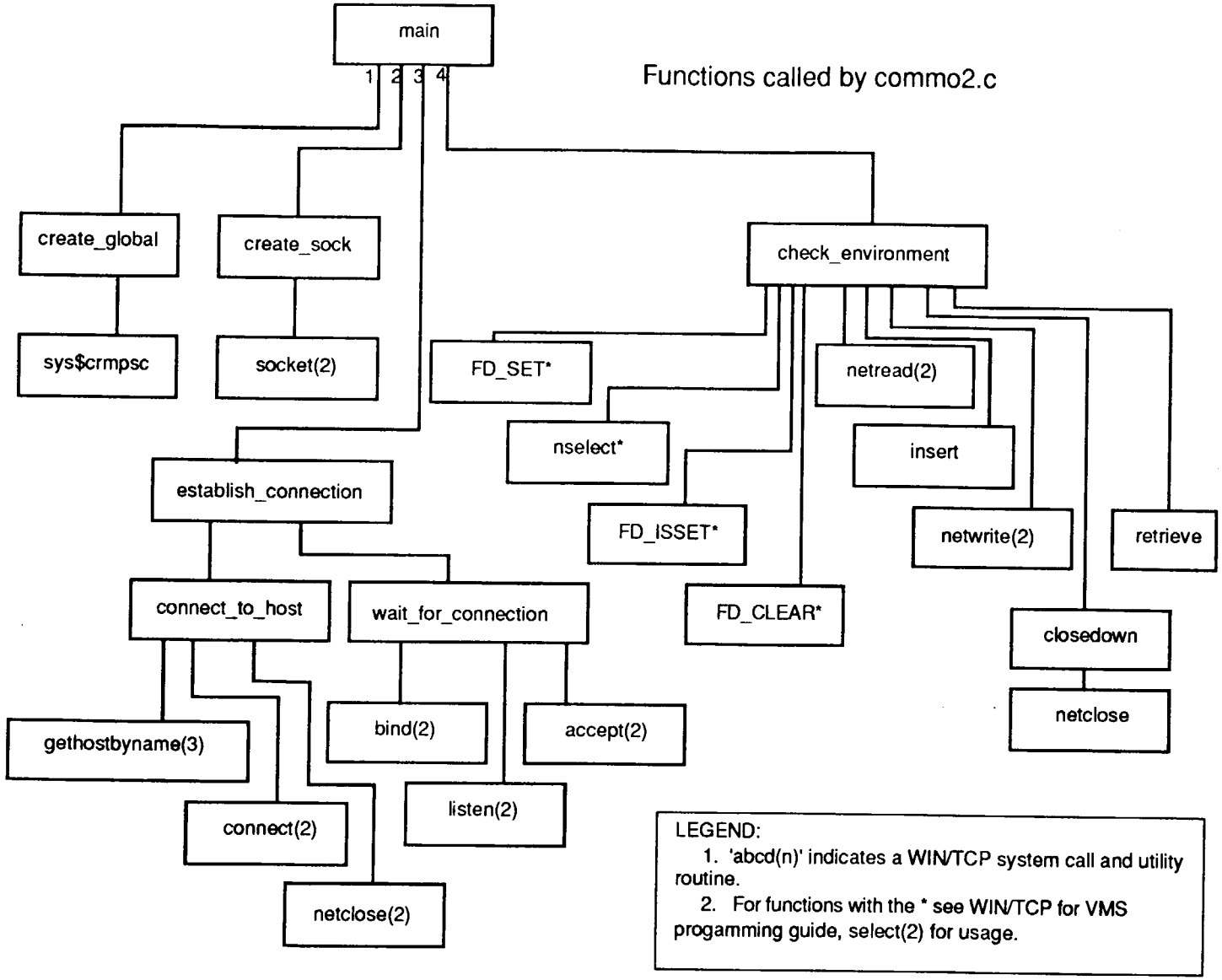

Figure 3. Block diagram of the main function commo2.

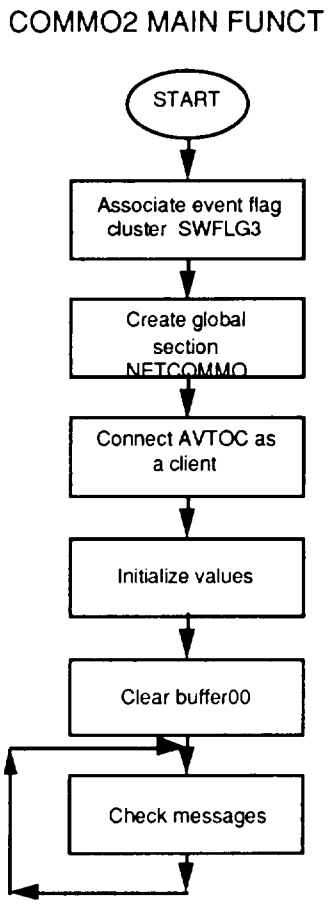

Figure 4. Flowchart of the main function commo2.

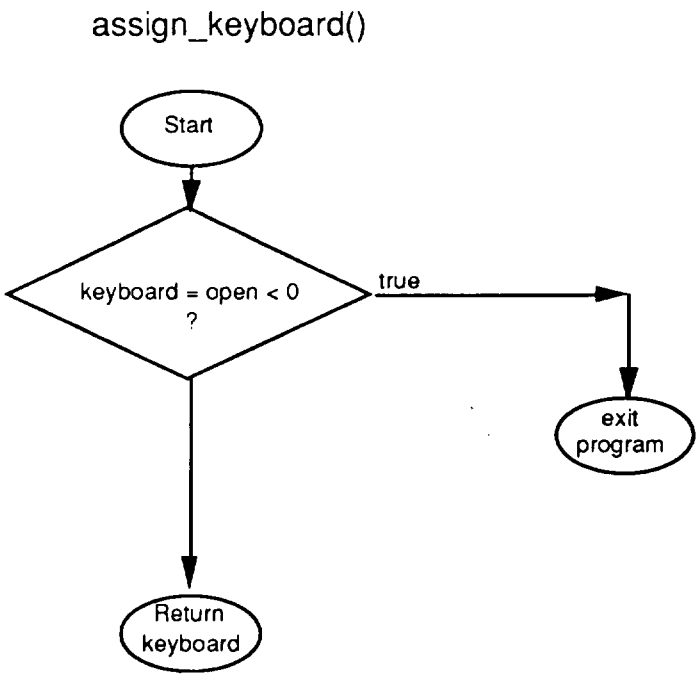

Figure 5. Flowchart of the function assign_keyboard.

FUNCTION CLOSEDOWN

This function de-assigns the channel for the socket. It is called from function **check_environment** when an error occurs while the program is writing to the net.

The argument passed to this function is

message - Character string containing the message to be displayed by the function.

The following TCP/IP function is called

netclose.

The flowchart for the function **closedown** is shown in Figure 6.

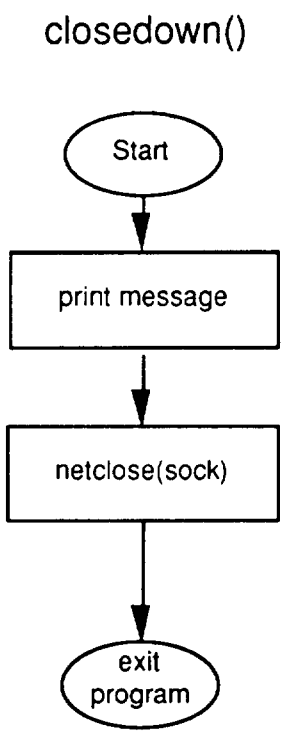

Figure 6. Flowchart of the function closedown.

FUNCTION CONNECT_TO_HOST

This function starts by calling a TCP/IP function **gethostbyname** to get the host's name of the remote server for a client type connection. It establishes connection with the AVTOC server by connecting to a socket and returns the socket number. It uses the Internet family address and a port number. This function is called in the function **establish_connection**.

The arguments passed to this function are

sock - Integer. Socket number to the AVTOC.

hostname - Character string. Host to which this process is going to connect.

port - Integer. Number assigned to the socket structure.

The following TCP/IP functions are called

gethostbyname, bzero, bcopy, htons, connect.

The flowchart for the function **connect_to_host** is shown in Figure 7.

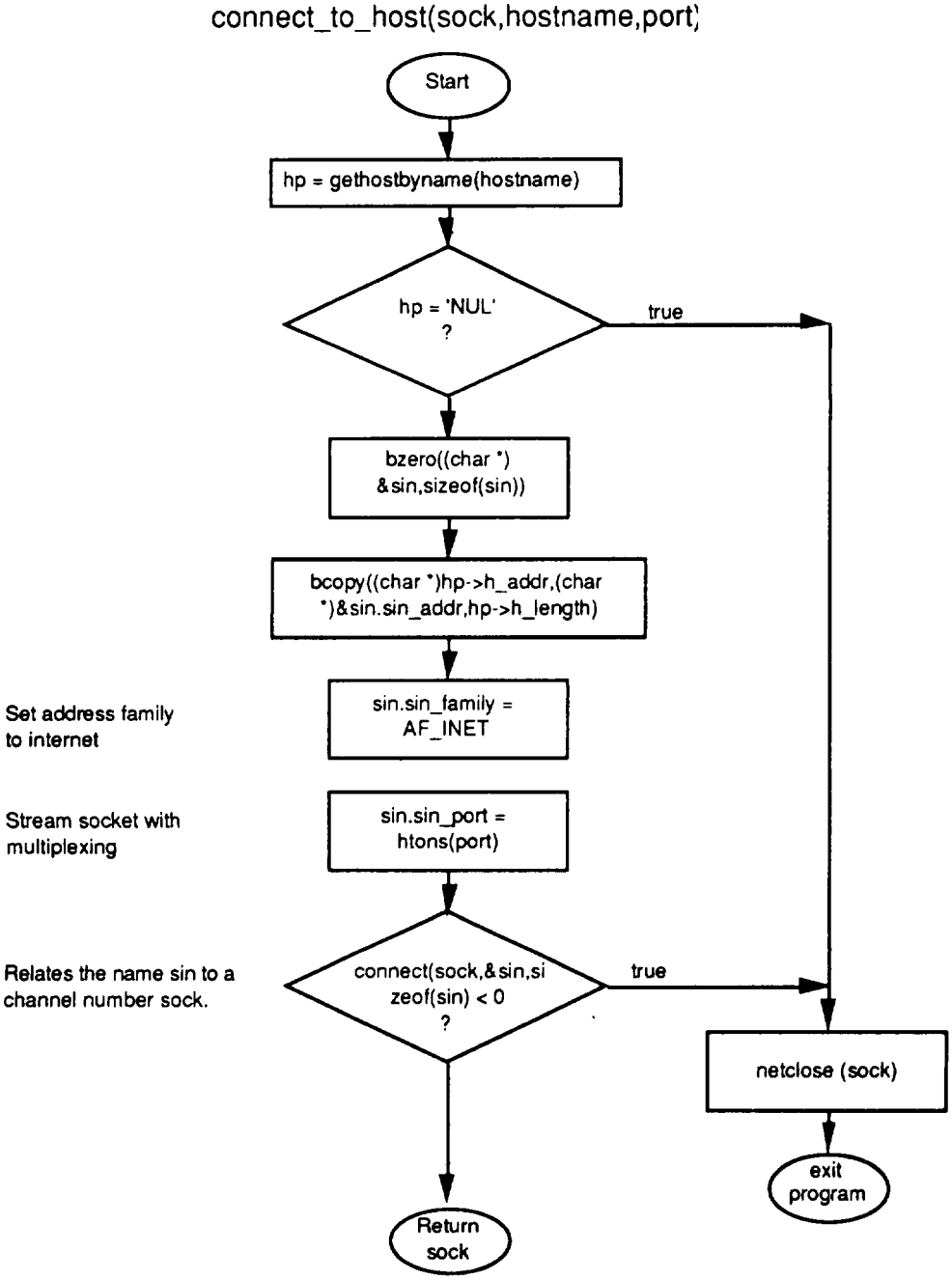

Figure 7. Flowchart of the function connect_to_host.

FUNCTION WAIT_FOR_CONNECTION

When the process acts as a server, it opens a socket to wait for a connection. This function uses the socket to accept a connection from the AVTOC process and assigns and returns a new socket for the AVTOC-VAX connection. This function is called in the function **establish_connection**.

The arguments passed to this function are

sock - Integer. Socket number to the AVTOC.
port - Integer. Port number for the AVTOC-VAX connection.

Other variables are

msgsock - Integer variable. New socket for the AVTOC-VAX connection.
length - Integer local variable. Size of the variable structure sin.
sin - Address structure containing TCP/IP information.

The following TCP/IP functions are called

bzero, htons, bind, listen, accept.

The flowchart for the function **wait_for_connection** is shown in Figure 8.

FUNCTION CHECK_USAGE

This function was called when the program was in the development stage to check if the process was intended to be a server or a client. The main function **commo2** was set to be called

commo2----------Server
commo2 hostname------Client

After the program was completed, it was decided that the VAX **commo2** process was to be client to the **bsd** process on the Silicon Graphics AVTOC node. The variable hostname is set within the program so that the **check_usage** call is eliminated.

The arguments passed to this function are

argc - Integer. Number of arguments entered at command line.
command - Character string. Command entered.

The flowchart for the function **check_usage** is shown in Figure 9.

FUNCTION CREATE_SOCK

This function is called to create a socket. It assigns a channel number to the socket with the Internet address family as domain, stream type, and TCP/IP protocol. This function returns the socket number used to listen for a connection when the TCP/IP function socket is called. It is called in the main function **commo2**.

sock - Integer. Socket number.

The flowchart for the function **create_sock** is shown in Figure 10.

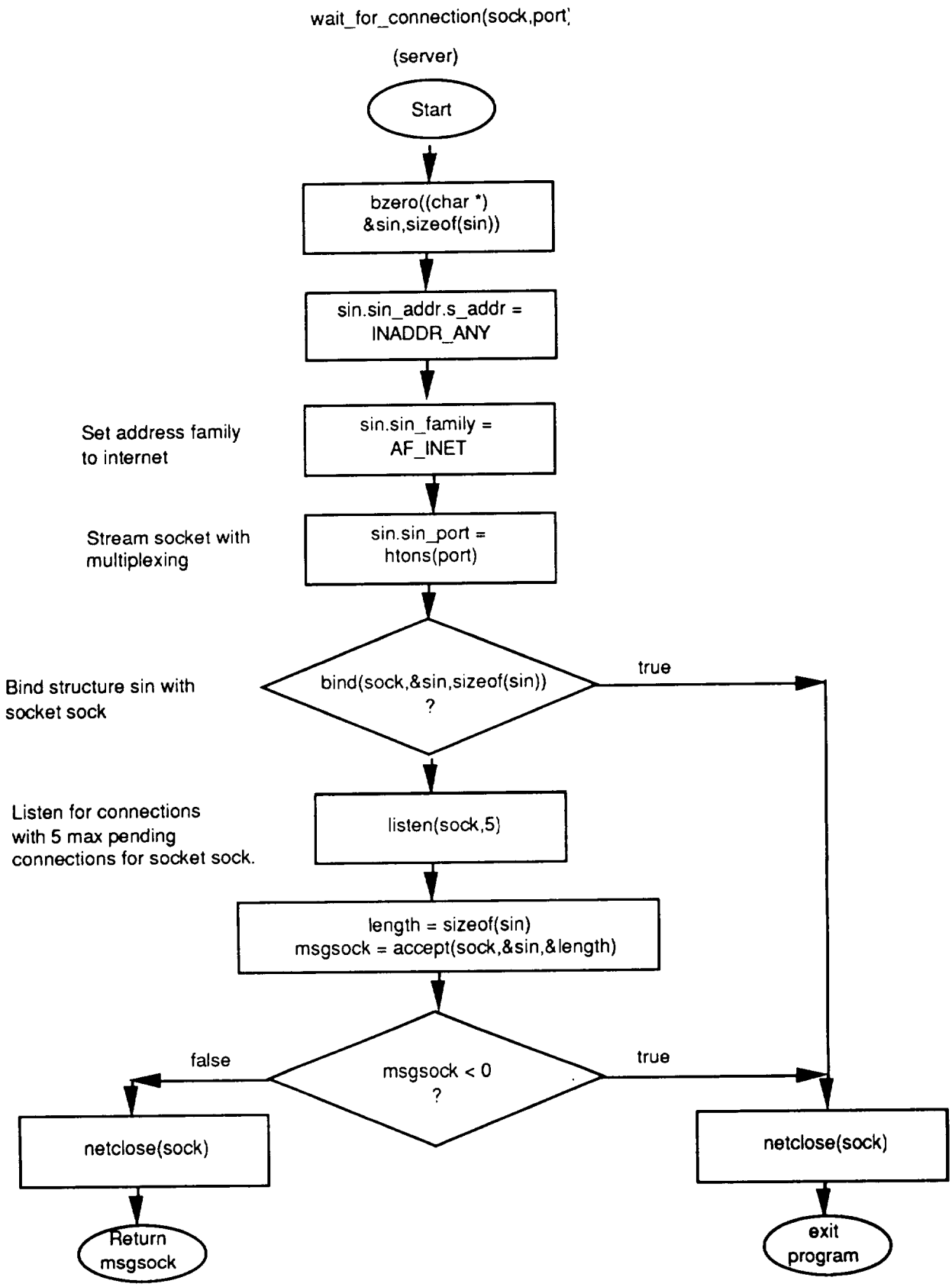

Figure 8. Flowchart of the function wait_for_connection.

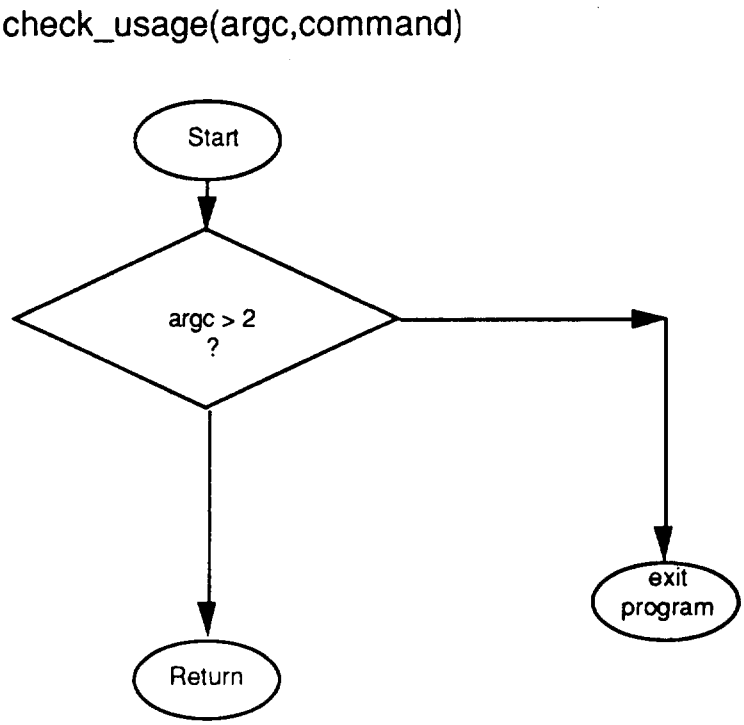

Figure 9. Flowchart of the function check_usage.

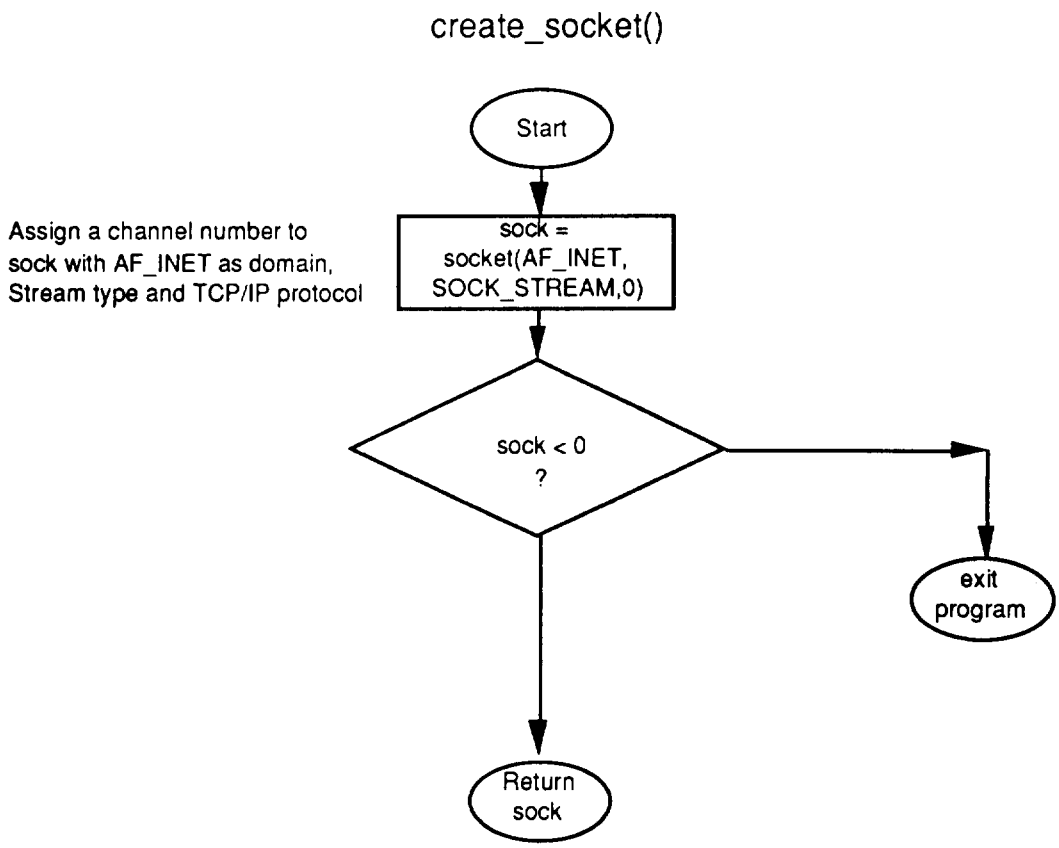

Figure 10. Flowchart of the function create_socket.

FUNCTION ESTABLISH_CONNECTION

This function calls the function **connect_to_host** if the process is a client or the function **wait_for_connection** if the process is a server. It returns a socket for the AVTOC-VAX connection in the variable "msgsock." This function is called in the main function **commo2**.

The arguments passed to this function are

sock - Integer. Socket number.
hostname - Character string. Host to which this process is going to connect.
argc - Integer. Number of arguments entered at command line.
port - Integer. Number assigned to the socket structure.
msgsock - Integer. New socket for the AVTOC-VAX connection.

The flowchart for the function **establish_connection** is shown in Figure 11.

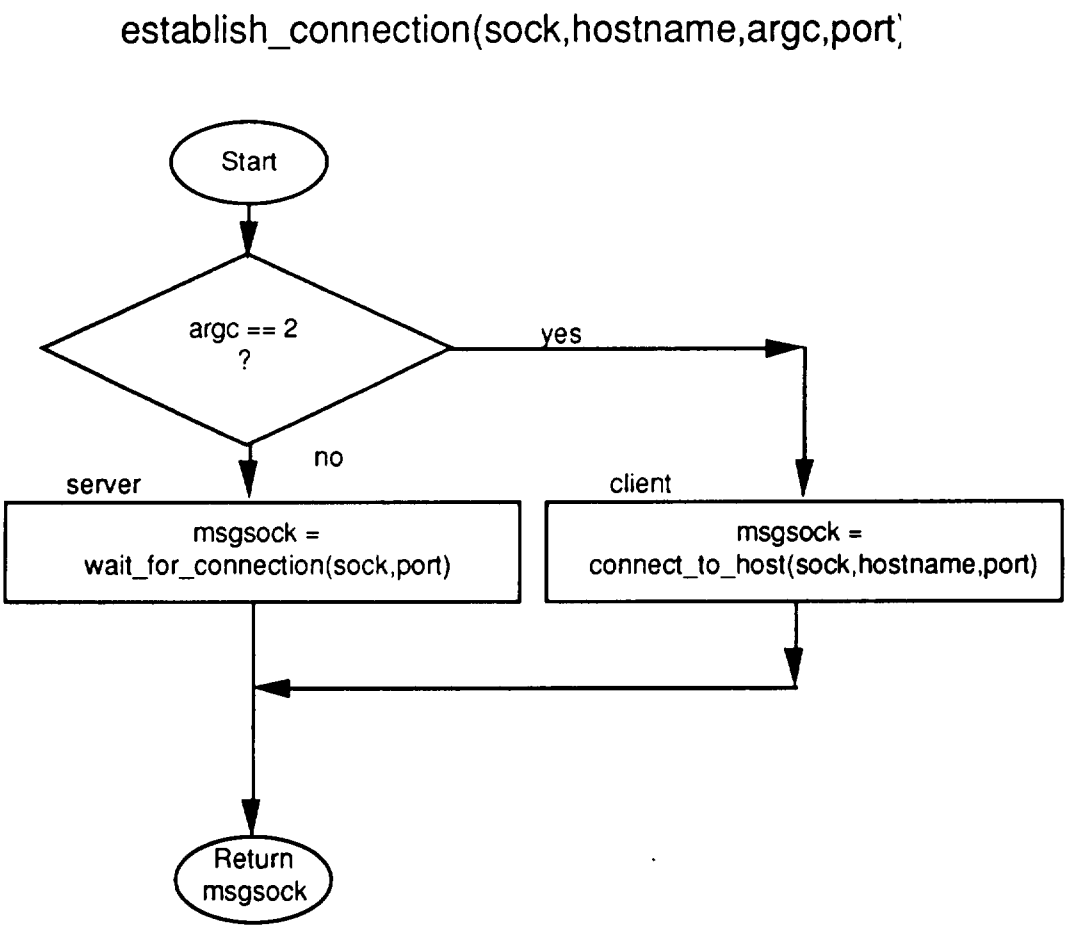

Figure 11. Flowchart of the function establish_connection.

FUNCTION INSERT

This function creates a linked list for the incoming messages from the AVTOC to be read by the helicopter pilot. It inserts the information at the end of the linked list. The "head_pointer" points to the first element of the list, "tail_pointer" points to the last element of the list. Each element on the list consists of an information field (a message) and an address field pointing to the next element. Before a message is inserted, an address is allocated for the new pointer.

This function is called in the function **check_environment** whenever there is an incoming message from the AVTOC.

The argument passed to this function is

buffer_arg - Pointer to a union "test2." Points to the last message sent from the AVTOC.

Other variables include

length - Integer local variable. Size of the variable buffer.
head_pointer - Global pointer to structure list. Points to the first element of the linked list.

tail_pointer - Global pointer to structure list. Points to the last element of the linked list.

first_insert - Integer global variable. Its value is 1 when the linked list is empty.

The C function **calloc** is called to allocate space for each element of the linked list.

The flowchart for the function **insert** is shown in Figure 12.

FUNCTION RETRIEVE

This function retrieves messages from the linked list when the pilot requests to read a message. It retrieves the information pointed to by the "head_pointer" in a first-in-first-out fashion.

This function is called in the function **check_environment** whenever the pilot requests to read a message by means of the communications display. There are no arguments to this function.

Other variables include

current_pointer - Pointer to a list structure. Points to the message that has been removed from the linked list.

The flowchart for the function **retrieve** is shown in Figure 13.

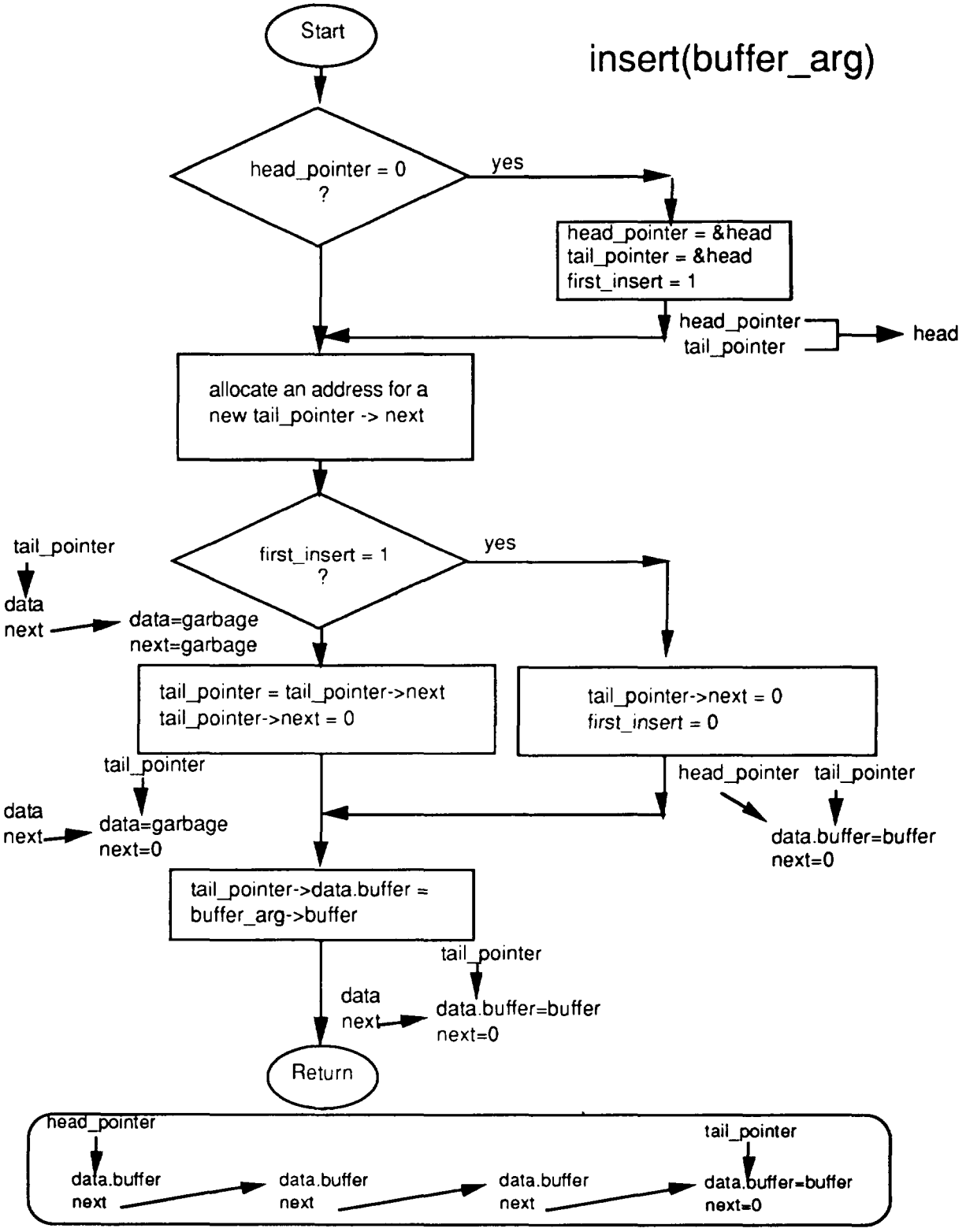

Figure 12. Flowchart of the function insert.

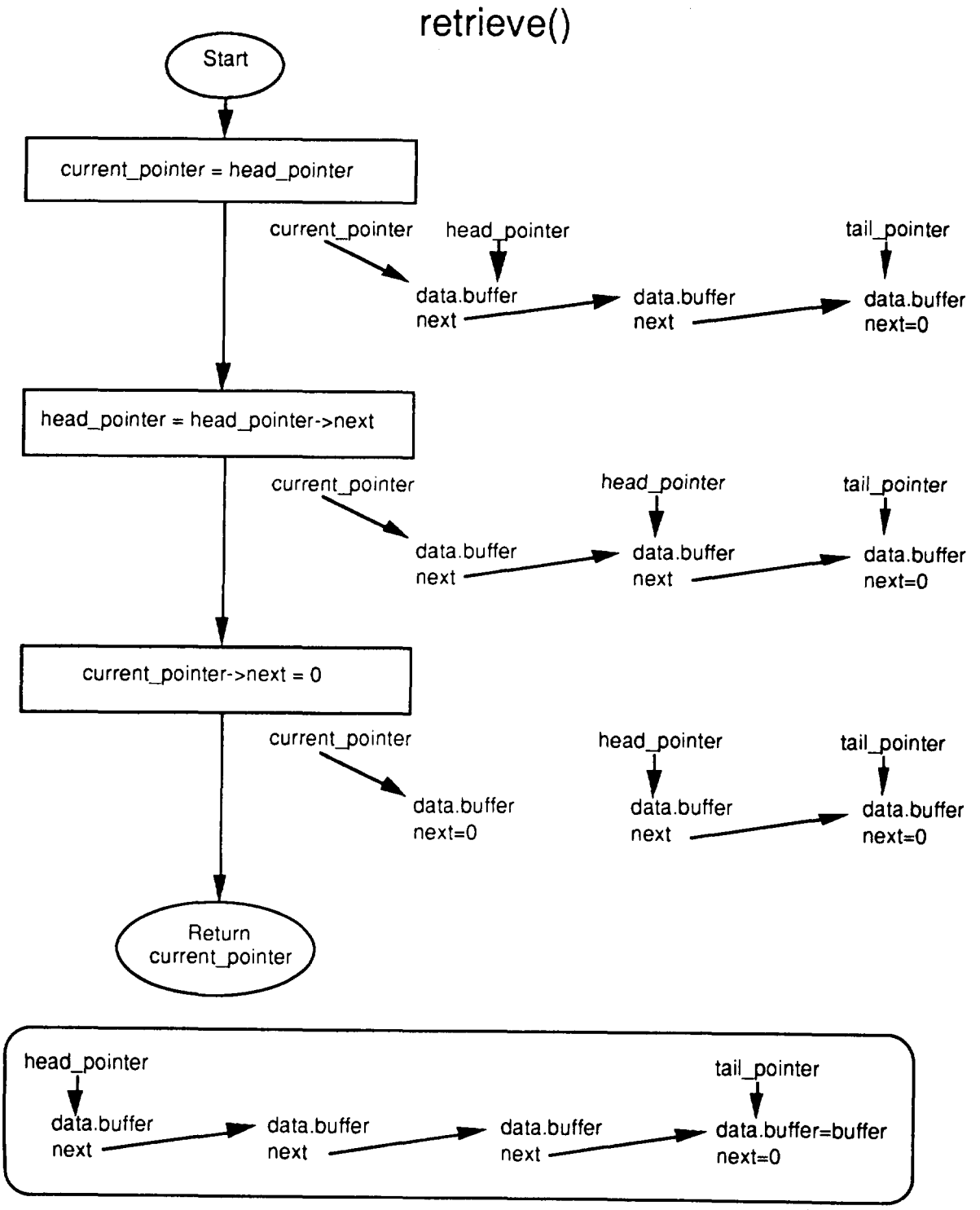

Figure 13. Flowchart of the function retrieve.

FUNCTION CHECK_ENVIRONMENT

This function is called in the main function **commo2**. It starts by checking for any activity on the communications channel. If there is activity (in this case from the AVTOC), it reads the information received. The first thing it checks for is a simulation control message indicating the start of the simulation. Once the simulation has started, check_environment looks for other messages such as data management, fire control, weapons control, and so forth. This function is responsible for setting event flag 120 to signal processes **commologic** and **startflights** when simulation has started.

Check_environment also checks for any messages to be sent from the **commologic** process and sends them if requested by the pilot.

This function also checks for a dropped track, generates and sends the drop track message to the AVTOC.

In the VAX memory, character bytes are arranged starting with the low order byte on the right as opposed to the Silicon Graphics memory where the bytes are arranged with the low order byte on the left. Because of this, VAX bytes are rearranged before they are transferred to the Silicon Graphics so they will be received in the right order. This is done before check_environment sends a message and after it receives a message.

This function calls functions **insert** and **retrieve**. The arguments passed to this function are

keyboard - Integer variable. Channel number for the keyboard as input device.

readchans - Structure fd_set. Channels to be checked for activity.

timeout - Structure timeval. Holds how long should the read instruction wait for a message (2 seconds).

buffer_arg - Pointer to a union test2. Points to the last message sent from the AVTOC.

Other variables include

nfound - Local integer. Its value is -1 if there is an error when returning from the nselect function. A positive or zero value represents the number of channels that have any activity when nselect was called.

i, cont1, cont2, cont3 - Local integer counter.

buflen - Local integer. The size of buffer.

status - Local integer. Status returned from a system service call.

done - Local integer. Conditional to end a loop when it is 1.

temp_buffer - Local character array. Temporary storage for the content of variable buffer. It is used when shifting bytes.

The following VAX/VMS system service routines are called

sys$waitfr, sys$clref, sys$clref.

This function calls the following TCP/IP functions

nselect, FD_CLR, FD_ISSET, FD_SET, netwrite, netread.

The flowchart for the function **check_environment** is shown in Figure 14.

FUNCTION PRINT_LIST

This function was called in the development stage of the program to print the linked list. It moves through the linked list using the current_pointer variable without altering it.

This function is not called during the simulation.

The flowchart for the function **print_list** is shown in Figure 15.

SUBROUTINE CREATE_GLOBAL

This FORTRAN subroutine called in the main routine **commo2** creates a global common section that will be shared by the FORTRAN processes **commologic, startflights**, and C process **commo2** during the simulation. A FORTRAN subroutine is a program unit consisting of a SUBROUTINE statement followed by a series of statements that define a computing procedure. A RETURN statement is used to return control to the calling program unit. The global section name is NETCOMMO; section size is one block mapped into NETCOMMO.DAT and to the common section named aanetcommo.

Notice that in program **commo2**, the variables shared with the FORTRAN processes are addressed with the common name aanetcommo. as prefix. The reason is that the variables of the FORTRAN common labeled aanetcommo share the same memory location as the members of the C language structure with variable name aanetcommo.

A special linker file is included in the Appendix C. The linker file is a command file that is run to link all object files together. This operation may be done interactively. Before a source-language program can run on VMS, it must be translated into object code and then linked. Each object module contains records that define its content and memory requirements to the linker.

check_environment(keyboard, readchans, timeout, buffer_arg)

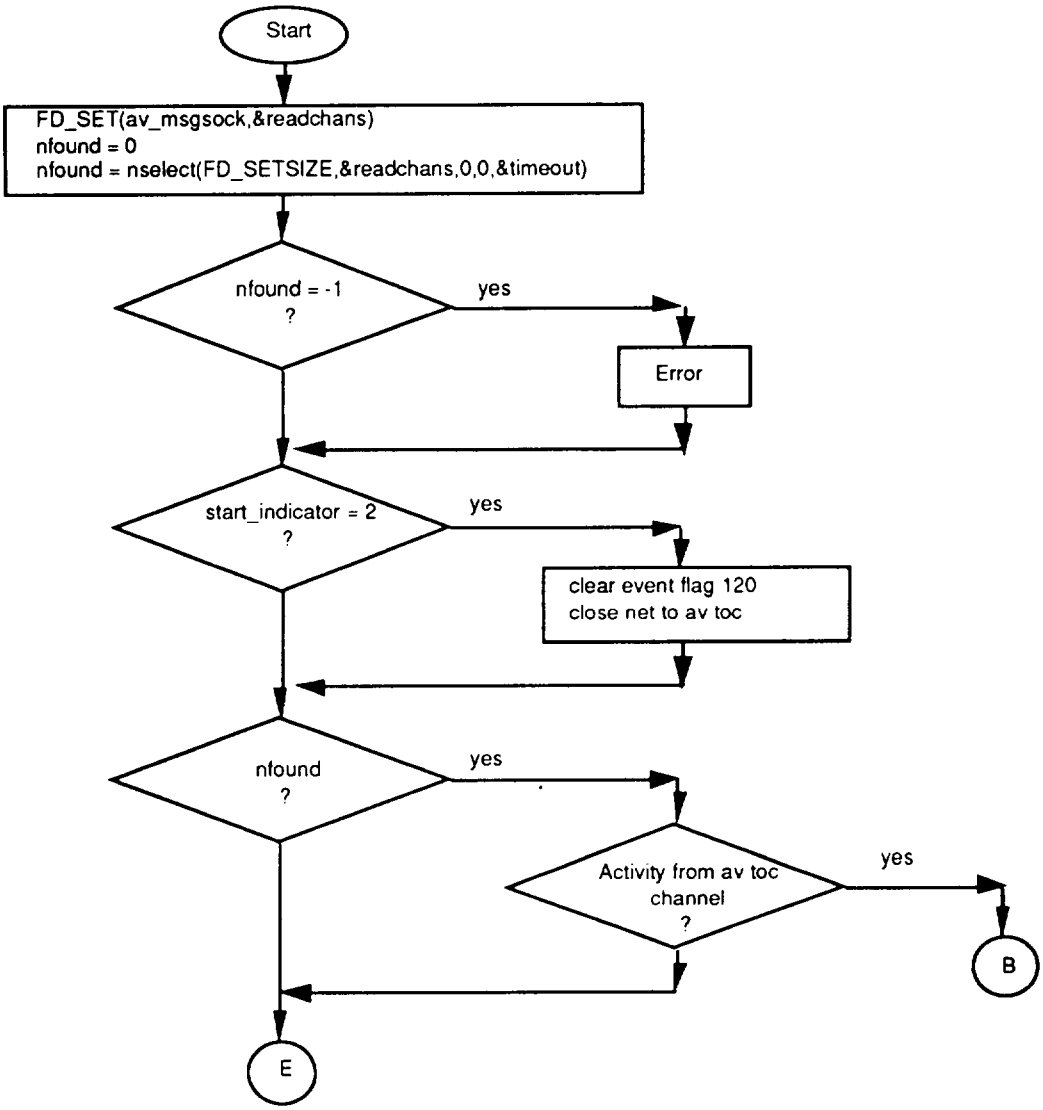

Figure 14. Flowchart of the function check_environment.

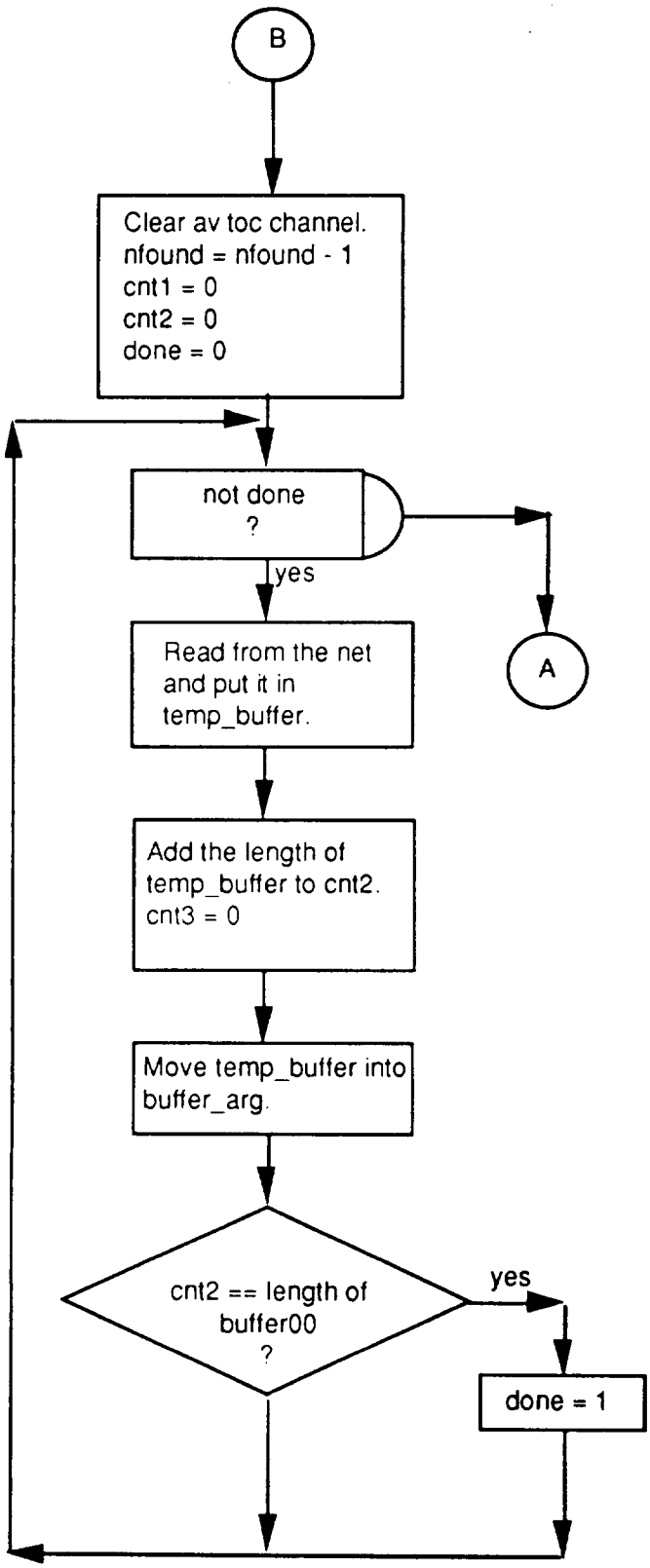

Figure 14. Flowchart of the function check_environment (cont'd).

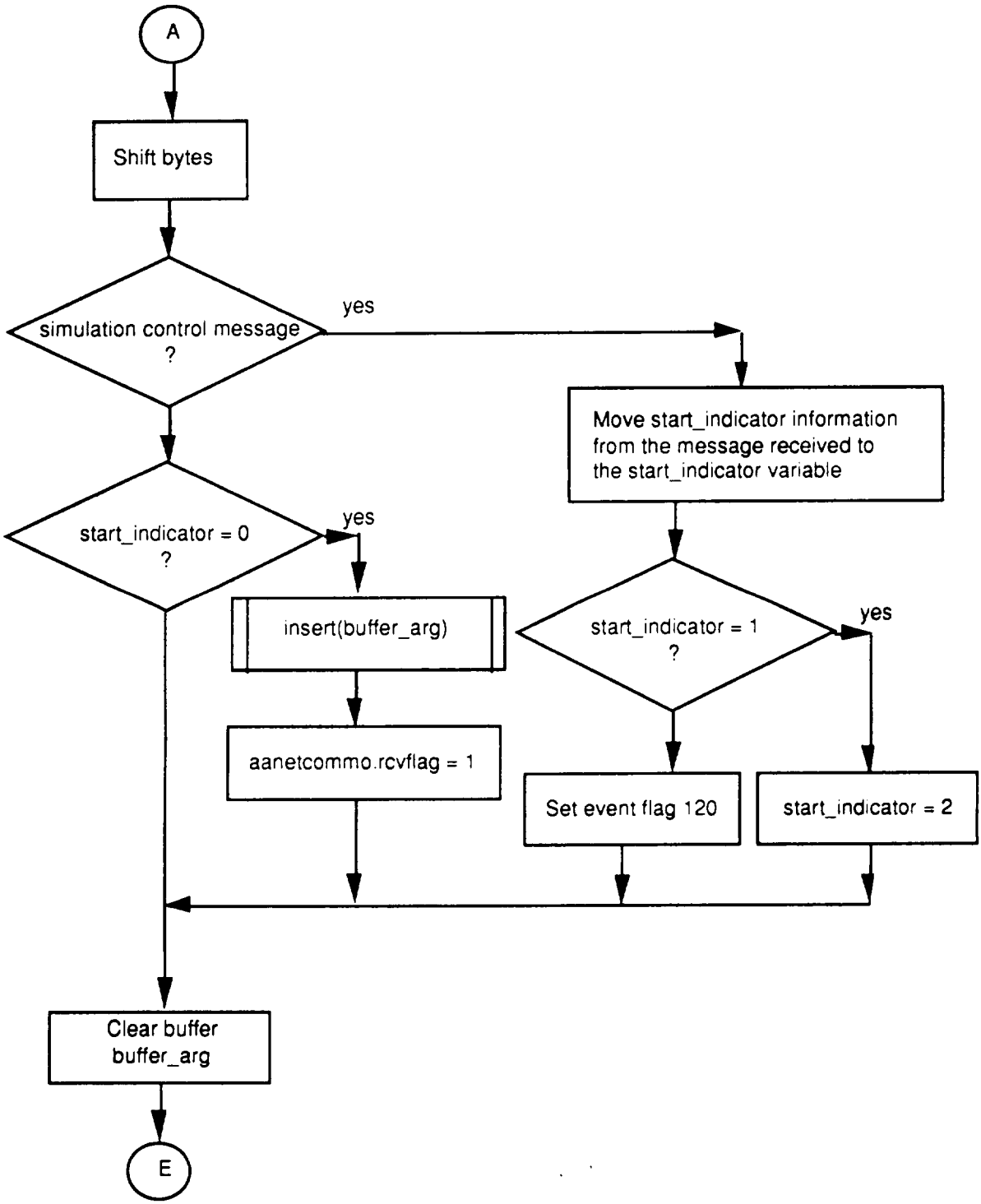

Figure 14. Flowchart of the function check_environment (cont'd).

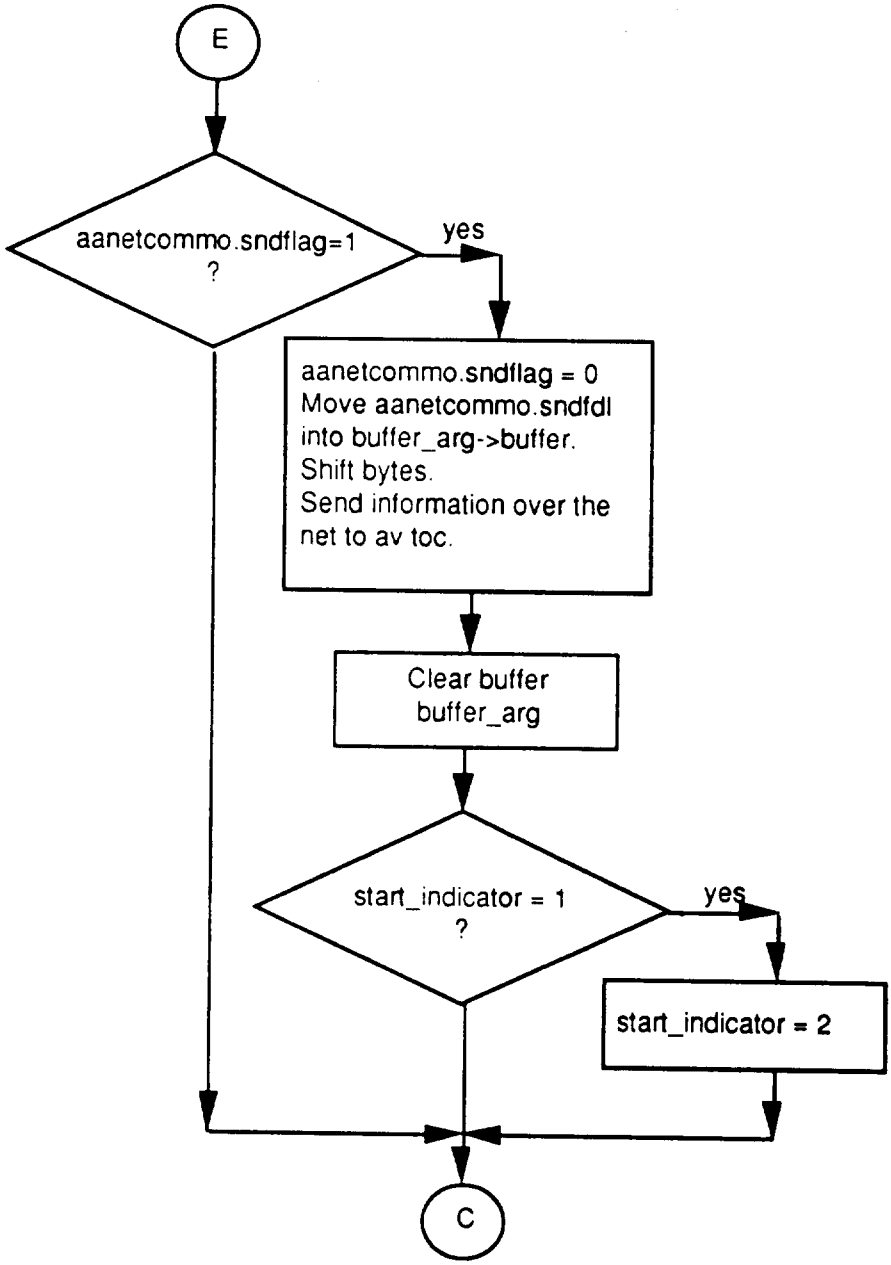

Figure 14. Flowchart of the function check_environment (cont'd).

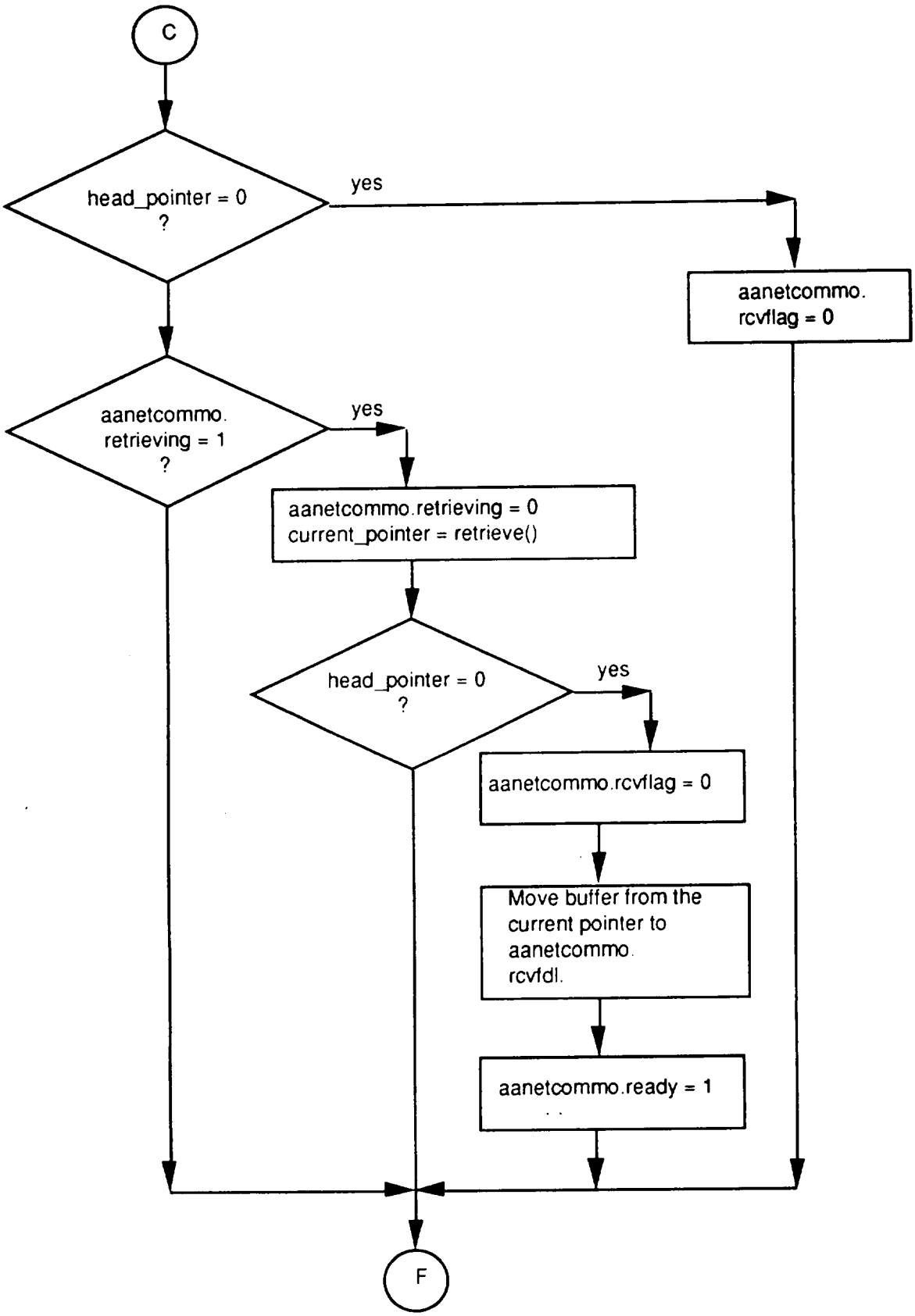

Figure 14. Flowchart of the function check_environment (cont'd).

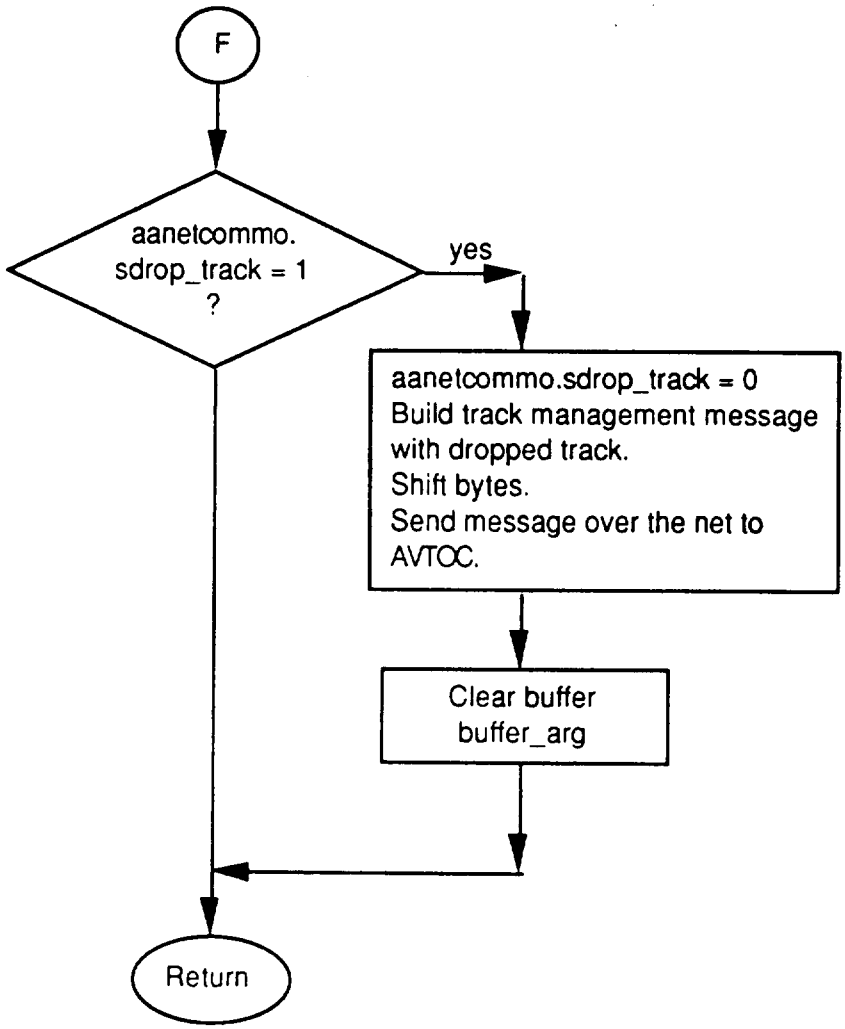

Figure 14. Flowchart of the function check_environment (cont'd).

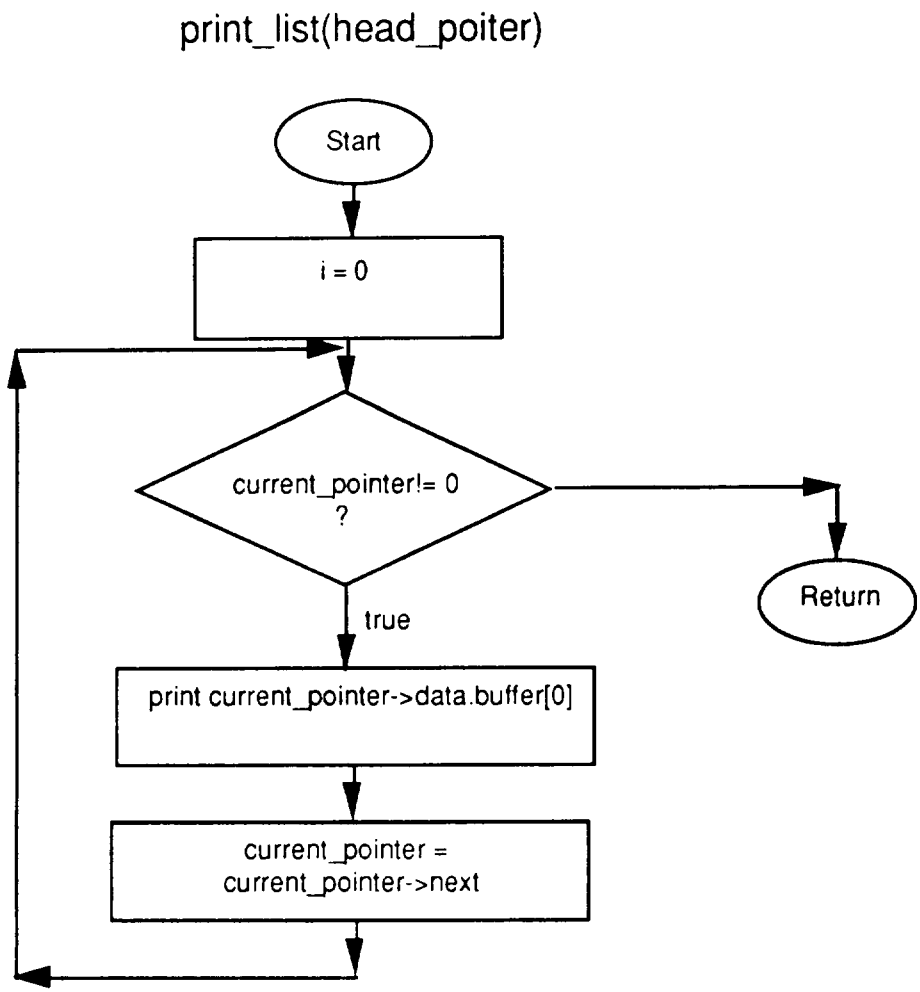

Figure 15. Flowchart of the function print_list.

The following variables are mapped and shared:

sndfdl - 12 bytes. Stores the message to be sent.

rcvfdl - 12 bytes. Stores the message received.

col_sw - Character string. Is set in the switch process with the value of the switch and is used by commologic process.

sdrop_track - Integer. Its value is one if there is a dropped track. This variable is set to one or zero in process **commo2**.

stn - Integer. Stores the track number of the most recent dropped track. It is used when sending the drop track message to the AVTOC.

ready - Integer. Its value is one when the process **commo2** is done retrieving and storing a message from the linked list.

rcvflag - Integer. Its value is one when the process **commo2** has finished adding a new message to the linked list. It is set to zero when the linked list is empty.

sndflag - Integer. Its value is one when the pilot has filled out a message to be sent to the AVTOC. This variable is set to one in the process commologic. It is set to zero in the process **commo2**.

retrieving - Integer. It is set to one by the process commologic when the pilot requests to read a message. It is set to zero by the process **commo2**.

Other variables include

get_chan - Integer local variable. External FORTRAN subroutine used when opening the global section file to get a channel for the section.

sec_chan - Integer local variable. Channel number for the section.

globsec - Character local variable. Common section name.

status - Integer local variable. Stores the status returned by the system service call.

sec_flags - Integer local variable. Flags used when creating the global section, secm_gbl, secm_wrt, sec$m_dzro.

maprange - Integer local array. Stores the address of the starting and ending point of the global section.

retadr - Integer local array. Stores the returning address of the section mapped.

The following system service routine is called

sys$crmpsc.

The flowchart for the function **create_global** is shown in Figure 16, and the source code is given in Appendix D.

CONCLUSIONS

The HELCAP simulation was demonstrated in July 1991. The communications software performed as expected.

The communications software arranged in functions makes it easier for the user to make modifications. These functions may be used separately in other applications since they are written in a general format.

Extra care should be taken when using shared global sections between FORTRAN and C programs. Variables that are not included in the global section may be updated unwillingly. When a C function that uses this approach is developed, it is recommended that a test program that shares a global section be created and run to set all variables used on this function to verify the actual values of these variables.

Using more than one computer to perform investigations of complex soldier-machine interfaces is an approach taken by HEL to support the wide range of tasks for the simulation needs and to allow for easy growth at a low initial cost.

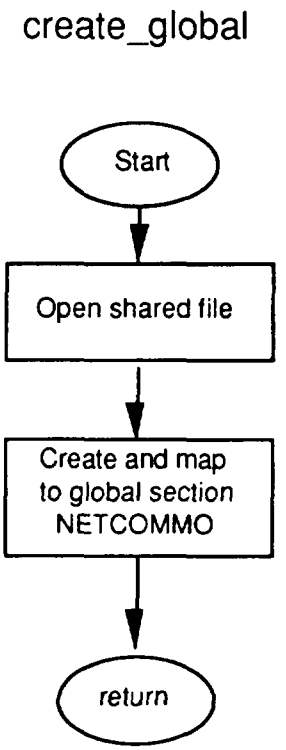

Figure 16. Flowchart of the function create_global.

REFERENCES

Digital Equipment Corporation (1989). VAX/VMS system services reference manual version 5.0 (or higher). Maynard, MA: Author.

The Wollongong Group, Inc. (1989). WIN/TCP for VMS programmer's guide version 5.0. Palo Alto, CA: Author.

Silicon Graphics Computer Systems, Inc. (1990). Network communications guide version 1.0. Mountain View, CA: Author.

Herald, G. (1992). HEL counter-air program pedestal-mounted stinger simulation (Technical Note 8-92). Aberdeen Proving Ground, MD: U.S. Army Human Engineering Laboratory.

U.S. Army Missile Command (1988). FAAD data link (FDL) technical interface design Plan (TIDP) (MIS 36264B). Redstone Arsenal, AL: Author.

Ware, N. (in press). HEL counter-air program aviation and air defense tactical operations centers simulation. Aberdeen Proving Ground, MD: U.S. Army Human Engineering Laboratory.

BIBLIOGRAPHY

Department of the Army (1987). Map reading and land navigation (Field Manual 21-26). Washington, D.C.: Author.

Digital Equipment Corporation (1989). Guide to VAX C version 5.0 (or higher). Maynard, MA: Author.

The Wollongong Group, Inc. (1989). WIN/TCP for VMS administrator's guide version 5.0. Palo Alto, CA: Author.

The Wollongong Group, Inc. (1989). WIN/TCP for VMS programming guide version 5.1. Palo Alto, CA: Author.

APPENDIX A

INCLUDE FILES

commo.h

```
/*.....................commo.h.....................*/
#include "types.h"
#include "socket.h"
#include "time.h"
#include "in.h"
#include "fcntl.h"
#include "field.h"
#include "netdb.h"
#define MY_PORT 1056
#define port5 1062  /* ADTOC */
#define port6 1064  /* PMS */
#define port3 1058  /* AVTOC */
```

errno.h

```
/*.....................errno.h.....................*/

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)errno.h 7.1 (Berkeley) 6/4/86
 */

/*
 * Error codes
 */

#define     EPERM       1           /* Not owner */
#define     ENOENT              2           /* No such file or directory */
#define     ESRCH       3           /* No such process */
#define     EINTR       4           /* Interrupted system call */
#define     EIO         5           /* I/O error */
#define     ENXIO       6           /* No such device or address */
#define     E2BIG       7           /* Arg list too long */
#define     ENOEXEC             8           /* Exec format error */
#define     EBADF       9           /* Bad file number */
#define     ECHILD              10          /* No children */
#define     EAGAIN              11          /* No more processes */
#define     ENOMEM              12          /* Not enough core */
#define     EACCES              13          /* Permission denied */
#define     EFAULT              14          /* Bad address */
#define     ENOTBLK             15          /* Block device required */
#define     EBUSY       16          /* Mount device busy */
#define     EEXIST              17          /* File exists */
#define     EXDEV       18          /* Cross-device link */
#define     ENODEV              19          /* No such device */
#define     ENOTDIR             20          /* Not a directory*/
#define     EISDIR              21          /* Is a directory */
#define     EINVAL              22          /* Invalid argument */
#define     ENFILE              23          /* File table overflow */
#define     EMFILE              24          /* Too many open files */
#define     ENOTTY              25          /* Not a typewriter */
#define     ETXTBSY             26          /* Text file busy */
#define     EFBIG       27          /* File too large */
#define     ENOSPC              28          /* No space left on device */
#define     ESPIPE              29          /* Illegal seek */
#define     EROFS       30          /* Read-only file system */
#define     EMLINK              31          /* Too many links */
#define     EPIPE       32          /* Broken pipe */

/* math software */
#define     EDOM        33          /* Argument too large */
#define     ERANGE              34          /* Result too large */

/* non-blocking and interrupt i/o */
#define     EWOULDBLOCK 35          /* Operation would block */
#define     EDEADLK             EWOULDBLOCK /* ditto */
#define     EINPROGRESS 36          /* Operation now in progress */
#define     EALREADY    37          /* Operation already in progress */
```

```
/* ipc/network software */

      /* argument errors */
#define     ENOTSOCK    38          /* Socket operation on non-socket */
#define     EDESTADDRREQ      39          /* Destination address required */
#define     EMSGSIZE    40          /* Message too long */
#define     EPROTOTYPE  41          /* Protocol wrong type for socket */
#define     ENOPROTOOPT 42          /* Protocol not available */
#define     EPROTONOSUPPORT   43          /* Protocol not supported */
#define     ESOCKTNOSUPPORT   44          /* Socket type not supported */
#define     EOPNOTSUPP  45          /* Operation not supported on socket */
#define     EPFNOSUPPORT      46          /* Protocol family not supported */
#define     EAFNOSUPPORT      47          /* Address family not supported by
protocol family */
#define     EADDRINUSE  48          /* Address already in use */
#define     EADDRNOTAVAIL     49          /* Can't assign requested address */

      /* operational errors */
#define     ENETDOWN    50          /* Network is down */
#define     ENETUNREACH 51          /* Network is unreachable */
#define     ENETRESET   52          /* Network dropped connection on reset */
#define     ECONNABORTED      53          /* Software caused connection abort
*/
#define     ECONNRESET  54          /* Connection reset by peer */
#define     ENOBUFS           55          /* No buffer space available */
#define     EISCONN           56          /* Socket is already connected */
#define     ENOTCONN    57          /* Socket is not connected */
#define     ESHUTDOWN   58          /* Can't send after socket shutdown */
#define     ETOOMANYREFS      59          /* Too many references: can't splice
*/
#define     ETIMEDOUT   60          /* Connection timed out */
#define     ECONNREFUSED      61          /* Connection refused */

      /* */
#define     ELOOP       62          /* Too many levels of symbolic links */
#define     ENAMETOOLONG      63          /* File name too long */

/* should be rearranged */
#define     EHOSTDOWN   64          /* Host is down */
#define     EHOSTUNREACH      65          /* No route to host */
#define     ENOTEMPTY   66          /* Directory not empty */

/* quotas & mush */
#define     EPROCLIM    67          /* Too many processes */
#define     EUSERS            68          /* Too many users */
#define     EDQUOT            69          /* Disc quota exceeded */
/* Network File System */
#define     ESTALE            70          /* Stale NFS file handle */
#define     EREMOTE           71          /* Too many levels of remote in path
*/
```

field.h

```
/*.....................field.h.....................*/
/*.....................................................................
*/
/*  File name:  field.h                                  Computer:  VAXLAB
*/
/*  Author:  Maria C. Lopez                              Date:  05-02-1991
*/
/*  Description:  This is an include file to commo.c.  This file sets up the
*/
/*  field declaration for the messages for the HELCAP demo.  The declarations
*/
/*  on this file are similar to the file on the IRIS machine that is going to
*/
/*  interchange messages except backwards because of the way the VAX
represent*/
/*  values in memory compare to the IRIS.
*/
/*  Directory:  CANCER::[lopez.helcap]
*/
/*.....................................................................
*/
      union test2 {
        char buffer[12];
      struct f1 { /*Data management*/
        unsigned spare4:16;
        unsigned spare3:16;
        unsigned spare2:32;
        unsigned spare1:4;
        unsigned sourceid:8;
        unsigned reference:4;
        unsigned segment:4;
        unsigned action:5;
        unsigned machine:1;
        unsigned sublabel:6;
      } buffer1;
      struct f4 { /*Track Management*/
        unsigned spare4:8;
        unsigned sourceid:8;
        unsigned idtrack:2;
        unsigned raidsize:2;
        unsigned astract:12;
        unsigned source:8;
        unsigned spare2:1;
        unsigned external:4;
        unsigned extract:19;
        unsigned action:4;
        unsigned pritrack:12;
        unsigned airclass:2;
        unsigned idevent:6;
        unsigned syssource:1;
        unsigned spare1:1;
        unsigned sublabel:6;
      } buffer4;
      struct f5 { /*Unit Operation Report*/
        unsigned spare4:16;
        unsigned spares:2;
        unsigned sourceid:8;
```

```
    unsigned point:2;
    unsigned nato:4;
    unsigned armor:2;
    unsigned explosive:4;
    unsigned proximity:4;
    unsigned chaparral:4;
    unsigned equiread:18;
    unsigned utype:4;
    unsigned stinger:5;
    unsigned trepair:2;
    unsigned nonsincgars:2;
    unsigned operstat:2;
    unsigned state:2;
    unsigned unit:8;
    unsigned machine:1;
    unsigned sublabel:6;
  } buffer5;

  struct f12 {      /*Weapon Control Order*/
    unsigned spare4:16;
    unsigned spare:2;
    unsigned sourceid:8;
    unsigned targetline:6;
    unsigned size:8;
    unsigned psi:1;
    unsigned refnumber:7;
    unsigned mresponse:2;
    unsigned alertord:2;
    unsigned adwarning:2;
    unsigned rotary:2;
    unsigned fixed:2;
    unsigned dactminute:6;
    unsigned dacthour:5;
    unsigned minute:6;
    unsigned hour:5;
    unsigned georefnum:9;
    unsigned machine:1;
    unsigned sublabel:6;
  } buffer12;

  struct f13 {      /*Enemy Activity Report*/
    unsigned spare4:16;
    unsigned spare3:2;
    unsigned sourceid:8;
    unsigned direction:3;
    unsigned size:3;
    unsigned minute:6;
    unsigned hour:5;
    unsigned day:5;
    unsigned y:13;
    unsigned spare2:1;
    unsigned target:2;
    unsigned x:13;
    unsigned spare1:1;
    unsigned activity:2;
    unsigned unit:4;
    unsigned weapon:4;
    unsigned mobility:1;
```

```
  unsigned machine:1;
  unsigned sublabel:6;
} buffer13;

struct f15 {      /*Movement Order*/
  unsigned spare4:16;
  unsigned spare2:8;
  unsigned sourceid:8;
  unsigned spare1:4;
  unsigned purpose:2;
  unsigned reponse:2;
  unsigned day:1;
  unsigned ordrefnum:7;
  unsigned suid:8;
  unsigned mission:2;
  unsigned relminute:6;
  unsigned relhour:5;
  unsigned minute:6;
  unsigned hour:5;
  unsigned georefnum:9;
  unsigned machine:1;
  unsigned sublabel:6;
} buffer15;

struct f16 {      /*Kill Report*/
  unsigned spare4:16;
  unsigned spare1:16;
  unsigned spare2:20;
  unsigned sourceid:8;
  unsigned missexp:4;
  unsigned roundexp:10;
  unsigned misskill:5;
  unsigned rotkill:5;
  unsigned fixkill:5;
  unsigned machine:1;
  unsigned sublabel:6;
}buffer16;

struct f18 {      /*Unit Location Report*/
  unsigned spare4:5;
  unsigned sourceid:8;
  unsigned heading:9;   /*degrees*/
  unsigned speed:10;    /*knots*/
  unsigned altitude:16; /*feet*/
  unsigned y:16;
  unsigned x:16;
  unsigned source:8;
  unsigned spare1:1;
  unsigned machine:1;
  unsigned sublabel:6;
} buffer18;

struct f26 {      /*Simulation Control*/
  unsigned spare4:16;
  unsigned spare3:16;
  unsigned spare2:32;
  unsigned spare1:16;
  unsigned sourceid:8;
```

```
    unsigned start_indicator:1;
    unsigned machine:1;
    unsigned sublabel:6;
  } buffer26;

  struct f29 {      /*Fire Control Unit*/
    unsigned spare5:8;
    unsigned sourceid:8;
    unsigned spare4:3;
    unsigned y:13;
    unsigned response:2;
    unsigned spare3:1;
    unsigned x:13;
    unsigned targetalt:7;
    unsigned spare2:1;
    unsigned source:8;
    unsigned action:4;
    unsigned targetrack:12;
    unsigned adduid:8;
    unsigned spare1:1;
    unsigned machine:1;
    unsigned sublabel:6;
  } buffer29;

  struct f60 {      /*Situation Report-Aviation*/
    unsigned spare5:16;
    unsigned spare3:8;
    unsigned sourceid:8;
    unsigned fuel:10;
    unsigned rounds:7;
    unsigned spare2:15;
    unsigned spare4:5;
    unsigned airgnd:4;
    unsigned airair:4;
    unsigned rockets:4;
    unsigned status:2;
    unsigned spare1:6;
    unsigned machine:1;
    unsigned sublabel:6;
  } buffer60;

  struct f61 {      /*Kill Report Aviation*/
    unsigned spare4:16;
    unsigned spare2:6;
    unsigned wheelkill:5;
    unsigned adkill:5;
    unsigned spare1:24;
    unsigned sourceid:8;
    unsigned apckill:5;
    unsigned tankkill:5;
    unsigned misskill:5;
    unsigned rotkill:5;
    unsigned fixkill:5;
    unsigned machine:1;
    unsigned sublabel:6;
  }buffer61;

  }buffer00;
```

```
/*.....................................................................*/
      struct list {
       union test2 data;
       struct list *next;
      };

      struct list *tail_pointer;
      struct list *head_pointer;
      struct list *current_pointer;
      struct list head;
      struct list tail;
      int first_insert = 1;
```

in.h

```
/*.....................in.h.....................*/

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)in.h    7.1 (Berkeley) 6/5/86
 */

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981.
 */

/*
 * Protocols
 */
#define     IPPROTO_IP          0           /* dummy for IP */
#define     IPPROTO_ICMP              1           /* control message protocol */
#define     IPPROTO_GGP         2           /* gateway^2 (deprecated) */
#define     IPPROTO_TCP         6           /* tcp */
#define     IPPROTO_EGP         8           /* exterior gateway protocol */
#define     IPPROTO_PUP         12          /* pup */
#define     IPPROTO_UDP         17          /* user datagram protocol */
#define     IPPROTO_IDP         22          /* xns idp */

#define     IPPROTO_RAW         255         /* raw IP packet */
#define     IPPROTO_MAX         256

#ifndef KERNEL
/*
 * Port/socket numbers: network standard functions
 */
#define     IPPORT_ECHO         7
#define     IPPORT_DISCARD            9
#define     IPPORT_SYSTAT             11
#define     IPPORT_DAYTIME            13
#define     IPPORT_NETSTAT            15
#define     IPPORT_FTP          21
#define     IPPORT_TELNET             23
#define     IPPORT_SMTP         25
#define     IPPORT_TIMESERVER   37
#define     IPPORT_NAMESERVER   42
#define     IPPORT_WHOIS              43
#define     IPPORT_MTP          57

/*
 * Port/socket numbers: host specific functions
 */
#define     IPPORT_TFTP         69
#define     IPPORT_RJE          77
#define     IPPORT_FINGER             79
#define     IPPORT_TTYLINK            87
#define     IPPORT_SUPDUP             95

/*
```

```
 * UNIX TCP sockets
 */
#define     IPPORT_EXECSERVER 512
#define     IPPORT_LOGINSERVER      513
#define     IPPORT_CMDSERVER  514
#define     IPPORT_EFSSERVER  520

/*
 * UNIX UDP sockets
 */
#define     IPPORT_BIFFUDP          512
#define     IPPORT_WHOSERVER  513
#define     IPPORT_ROUTESERVER      520   /* 520+1 also used */
#endif

/*
 * Ports < IPPORT_RESERVED are reserved for
 * privileged processes (e.g. root).
 * Ports > IPPORT_USERRESERVED are reserved
 * for servers, not necessarily privileged.
 */
#define     IPPORT_RESERVED         1024
#define     IPPORT_USERRESERVED     5000

/*
 * Link numbers
 */
#define     IMPLINK_IP        155
#define     IMPLINK_LOWEXPER  156
#define     IMPLINK_HIGHEXPER 158

/*
 * Internet address (a structure for historical reasons)
 */
struct in_addr {
      u_long s_addr;
};

/*
 * Definitions of bits in internet address integers.
 * On subnets, the decomposition of addresses to host and net parts
 * is done according to subnet mask, not the masks here.
 */
#define     IN_CLASSA(i)            (((long)(i) & 0x80000000) == 0)
#define     IN_CLASSA_NET           0xff000000
#define     IN_CLASSA_NSHIFT  24
#define     IN_CLASSA_HOST          0x00ffffff
#define     IN_CLASSA_MAX           128

#define     IN_CLASSB(i)            (((long)(i) & 0xc0000000) == 0x80000000)
#define     IN_CLASSB_NET           0xffff0000
#define     IN_CLASSB_NSHIFT  16
#define     IN_CLASSB_HOST          0x0000ffff
#define     IN_CLASSB_MAX           65536

#define     IN_CLASSC(i)            (((long)(i) & 0xc0000000) == 0xc0000000)
#define     IN_CLASSC_NET           0xffffff00
#define     IN_CLASSC_NSHIFT  8
```

```
#define     IN_CLASSC_HOST          0x000000ff

#define     INADDR_ANY        (u_long)0x00000000
#define     INADDR_BROADCAST  (u_long)0xffffffff      /* must be masked */

/*
 * Socket address, internet style.
 */
struct sockaddr_in {
      short sin_family;
      u_short     sin_port;
      struct      in_addr sin_addr;
      char  sin_zero[8];
};

/*
 * Options for use with [gs]etsockopt at the IP level.
 */
#define     IP_OPTIONS  1           /* set/get IP per-packet options */

#ifdef notdef
/*
 * Macros for number representation conversion.
 */
#define     ntohl(x)    (x)
#define     ntohs(x)    (x)
#define     htonl(x)    (x)
#define     htons(x)    (x)
#endif

u_short     ntohs(), htons();
u_long      ntohl(), htonl();

#ifdef KERNEL
extern      struct domain inetdomain;
extern      struct protosw inetsw[];
struct      in_addr in_makeaddr();
u_long      in_netof(), in_lnaof();
#endif

#ifndef     INCLUDE_NETDB_H
#define     INCLUDE_NETDB_H
/*
static char S_netdb.h [] = "@(#)netdb.h       3.1       23:24:15 - 87/07/27 ";
 */
```

netdb.h

```
/*.....................netdb.h.....................*/

/*
 * Copyright (c) 1980 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)netdb.h 5.7 (Berkeley) 5/12/86
 */

/*
 * Structures returned by network
 * data base library.  All addresses
 * are supplied in host order, and
 * returned in network order (suitable
 * for use in system calls).
 */
struct      hostent {
      char  *h_name;    /* official name of host */
      char  **h_aliases;      /* alias list */
      int   h_addrtype; /* host address type */
      int   h_length;   /* length of address */
      char  **h_addr_list;    /* list of addresses from name server */
#define     h_addr      h_addr_list[0]    /* address, for backward
compatiblity */
};

/*
 * Assumption here is that a network number
 * fits in 32 bits -- probably a poor one.
 */
struct      netent {
      char  *n_name;    /* official name of net */
      char  **n_aliases;      /* alias list */
      int   n_addrtype; /* net address type */
      int   n_net;            /* network # */
};

struct      servent {
      char  *s_name;    /* official service name */
      char  **s_aliases;      /* alias list */
      int   s_port;           /* port # */
      char  *s_proto;   /* protocol to use */
};

struct      protoent {
      char  *p_name;    /* official protocol name */
      char  **p_aliases;      /* alias list */
      int   p_proto;    /* protocol # */
};

struct rpcent {
      char  *r_name;    /* name of server for this rpc program */
      char  **r_aliases;      /* alias list */
      int   r_number;   /* rpc program number */
};
```

```
struct hostent    *gethostbyname(), *gethostbyaddr(), *gethostent();
struct netent     *getnetbyname(), *getnetbyaddr(), *getnetent();
struct servent    *getservbyname(), *getservbyport(), *getservent();
struct protoent   *getprotobyname(), *getprotobynumber(), *getprotoent();
struct rpcent     *getrpcbyname(), *getrpcbynumber(), *getrpcent();

/*
 * Error return codes from gethostbyname() and gethostbyaddr()
 */

extern  int h_errno;

#define     HOST_NOT_FOUND    1 /* Authoritive Answer Host not found */
#define     TRY_AGAIN   2 /* Non-Authoritive Host not found, or SERVERFAIL */
#define     NO_RECOVERY 3 /* Non recoverable errors, FORMERR, REFUSED, NOTIMP
*/
#define NO_ADDRESS      4 /* Valid host name, no address, look for MX record
*/

#endif      INCLUDE_NETDB_H

#define IPPORT_RESERVED 1024
#define PORT_ECHO 1056
#define PORT_FILE 1056
#define PORT_DISC 1056

#define INADDR_ANY      (u_long)0x00000000

struct in_addr {
      union {
            struct { unsigned char s_b1,s_b2,s_b3,s_b4;} S_un_b;
            struct { unsigned short s_w1,s_w2;} S_un_w;
            unsigned long S_addr;
      } S_un;
#define s_addr S_un.S_addr
#define s_host S_un.S_un_b.s_b2
#define s_net S_un.S_un_b.s_b1
#define s_imp S_un.S_un_w.s_w2
#define s_impno S_un.S_un_b.s_b4
#define s_lh S_un.S_un_b.s_b3
};
struct sockaddr_in {
      short sin_family;
      unsigned short sin_port;
      struct in_addr sin_addr;
      char sin_zero[8];
};
```

socket.h

```
/*.....................socket.h.....................*/

/*
 * Copyright (c) 1982,1985, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)socket.h      7.1 (Berkeley) 6/4/86
 */

/*
 * Definitions related to sockets: types, address families, options.
 */

/*
 * Types
 */
#define     SOCK_STREAM 1           /* stream socket */
#define     SOCK_DGRAM  2           /* datagram socket */
#define     SOCK_RAW    3           /* raw-protocol interface */
#define     SOCK_RDM    4           /* reliably-delivered message */
#define     SOCK_SEQPACKET    5           /* sequenced packet stream */

/*
 * Option flags per-socket.
 */
#define     SO_DEBUG    0x0001            /* turn on debugging info recording
*/
#define     SO_ACCEPTCONN     0x0002            /* socket has had listen() */
#define     SO_REUSEADDR      0x0004            /* allow local address reuse
*/
#define     SO_KEEPALIVE      0x0008            /* keep connections alive */
#define     SO_DONTROUTE      0x0010            /* just use interface
addresses */
#define     SO_BROADCAST      0x0020            /* permit sending of broadcast
msgs */
#define     SO_USELOOPBACK    0x0040            /* bypass hardware when
possible */
#define     SO_LINGER   0x0080            /* linger on close if data present
*/
#define     SO_OOBINLINE      0x0100            /* leave received OOB data in
line */
#define SO_NOUDPCKSUM   0x0200            /* Do not use cksums (UDP only) */

/*
 * Additional options, not kept in so_options.
 */
#define SO_SNDBUF 0x1001            /* send buffer size */
#define SO_RCVBUF 0x1002            /* receive buffer size */
#define SO_SNDLOWAT     0x1003            /* send low-water mark */
#define SO_RCVLOWAT     0x1004            /* receive low-water mark */
#define SO_SNDTIMEO     0x1005            /* send timeout */
#define SO_RCVTIMEO     0x1006            /* receive timeout */
#define     SO_ERROR    0x1007            /* get error status and clear */
#define     SO_TYPE           0x1008            /* get socket type */

/*
```

```
 * Structure used for manipulating linger option.
 */
struct      linger {
      int   l_onoff;          /* option on/off */
      int   l_linger;         /* linger time */
};

/*
 * Level number for (get/set)sockopt() to apply to socket itself.
 */
#define     SOL_SOCKET  0xffff            /* options for socket level */

/*
 * Address families.
 */
#define     AF_UNSPEC   0           /* unspecified */
#define     AF_UNIX           1           /* local to host (pipes, portals) */
#define     AF_INET           2           /* internetwork: UDP, TCP, etc. */
#define     AF_IMPLINK  3           /* arpanet imp addresses */
#define     AF_PUP            4           /* pup protocols: e.g. BSP */
#define     AF_CHAOS    5           /* mit CHAOS protocols */
#define     AF_NS       6           /* XEROX NS protocols */
#define     AF_NBS            7           /* nbs protocols */
#define     AF_ECMA           8           /* european computer manufacturers
*/
#define     AF_DATAKIT  9           /* datakit protocols */
#define     AF_CCITT    10          /* CCITT protocols, X.25 etc */
#define     AF_SNA            11          /* IBM SNA */
#define AF_DECnet 12          /* DECnet */
#define AF_DLI          13          /* Direct data link interface */
#define AF_LAT          14          /* LAT */
#define     AF_HYLINK   15          /* NSC Hyperchannel */
#define     AF_APPLETALK      16          /* Apple Talk */
#ifdef VAXVMS
#define     AF_RAWPACKET      17          /* raw packets (rp driver) */
#define     AF_EPACKET  18          /* ethernet packets (ep driver) */
#define AF_DBRIDGE      19          /* dbridge packets (db driver) */
#define AF_SIMBI  20          /* simpact BI raw interf (simbi) */

#define     AF_MAX            21
#else

#define     AF_MAX            17
#endif

/*
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
      u_short     sa_family;        /* address family */
      char  sa_data[14];            /* up to 14 bytes of direct address */
};

/*
 * Structure used by kernel to pass protocol
 * information in raw sockets.
 */
```

```
struct sockproto {
      u_short     sp_family;        /* address family */
      u_short     sp_protocol;            /* protocol */
};

/*
 * Protocol families, same as address families for now.
 */
#define     PF_UNSPEC   AF_UNSPEC
#define     PF_UNIX           AF_UNIX
#define     PF_INET           AF_INET
#define     PF_IMPLINK  AF_IMPLINK
#define     PF_PUP            AF_PUP
#define     PF_CHAOS    AF_CHAOS
#define     PF_NS       AF_NS
#define     PF_NBS            AF_NBS
#define     PF_ECMA           AF_ECMA
#define     PF_DATAKIT  AF_DATAKIT
#define     PF_CCITT    AF_CCITT
#define     PF_SNA            AF_SNA
#define PF_DECnet AF_DECnet
#define PF_DLI          AF_DLI
#define PF_LAT          AF_LAT
#define     PF_HYLINK   AF_HYLINK
#define     PF_APPLETALK      AF_APPLETALK
#ifdef VAXVMS
#define     PF_RAWPACKET      AF_RAWPACKET
#define     PF_EPACKET  AF_EPACKET
#define     PF_DBRIDGE  AF_DBRIDGE
#define     PF_SIMBI    AF_SIMBI
#endif

#define     PF_MAX            AF_MAX

/*
 * Maximum queue length specifiable by listen.
 */
#define     SOMAXCONN   5

/*
 * Message header for recvmsg and sendmsg calls.
 */
struct msghdr {
      caddr_t     msg_name;         /* optional address */
      int   msg_namelen;            /* size of address */
      struct      iovec *msg_iov;         /* scatter/gather array */
      int   msg_iovlen;       /* # elements in msg_iov */
      caddr_t     msg_accrights;          /* access rights sent/received */
      int   msg_accrightslen;
};

#define     MSG_OOB           0x1         /* process out-of-band data */
#define     MSG_PEEK    0x2         /* peek at incoming message */
#define     MSG_DONTROUTE     0x4         /* send without using routing tables
*/

#define     MSG_MAXIOVLEN     16
```

time.h

```
/*.....................time.h.....................*/

/*
 * Copyright (c) 1982 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)time.h  6.4 (Berkeley) 6/24/85
 */

/*
 * Structure returned by gettimeofday(2) system call,
 * and used in other calls.
 */
struct timeval {
      long  tv_sec;           /* seconds */
      long  tv_usec;    /* and microseconds */
};

struct timezone {
      int   tz_minuteswest;   /* minutes west of Greenwich */
      int   tz_dsttime; /* type of dst correction */
};
#define     DST_NONE    0     /* not on dst */
#define     DST_USA           1     /* USA style dst */
#define     DST_AUST    2     /* Australian style dst */
#define     DST_WET           3     /* western European dst */
#define     DST_MET           4     /* Middle European dst */
#define     DST_EET           5     /* eastern European dst */
#define     DST_CAN           6     /* Canada */

/*
 * Operations on timevals.
 *
 * NB: timercmp does not work for >= or <=.
 */
#define     timerisset(tvp)         ((tvp)->tv_sec || (tvp)->tv_usec)
#define     timercmp(tvp, uvp, cmp) \
      ((tvp)->tv_sec cmp (uvp)->tv_sec || \
       (tvp)->tv_sec == (uvp)->tv_sec && (tvp)->tv_usec cmp (uvp)->tv_usec)
#define     timerclear(tvp)         (tvp)->tv_sec = (tvp)->tv_usec = 0

/*
 * Names of the interval timers, and structure
 * defining a timer setting.
 */
#define     ITIMER_REAL 0
#define     ITIMER_VIRTUAL    1
#define     ITIMER_PROF 2

struct      itimerval {
      struct      timeval it_interval;    /* timer interval */
      struct      timeval it_value; /* current value */
};
#ifndef KERNEL
#include time
#endif
```

types.h

```
/*.....................types.h.....................*/

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *    @(#)types.h 7.1 (Berkeley) 6/4/86
 */

#ifndef _TYPES_
#define     _TYPES_
/*
 * Basic system types and major/minor device constructing/busting macros.
 */

/* major part of a device */
#define     major(x)    ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define     minor(x)    ((int)((x)&0377))

/* make a device number */
#define     makedev(x,y)      ((dev_t)(((x)<<8) | (y)))

typedef     unsigned char     u_char;
typedef     unsigned short    u_short;
typedef     unsigned int      u_int;
typedef     unsigned long     u_long;
typedef     unsigned short    ushort;          /* sys III compat */

#ifdef vax
typedef     struct      _physadr { int r[1]; } *physadr;
typedef     struct      label_t     {
      int   val[14];
} label_t;
#endif
typedef     struct      _quad { long val[2]; } quad;
typedef     long  daddr_t;
typedef     char *      caddr_t;
#ifndef __STAT
typedef     u_long      ino_t;
#endif
typedef     long  swblk_t;
#ifndef __STDDEF
#define __STDDEF
typedef     long  size_t;
#endif
#ifndef __TYPES
#define __TYPES
typedef     long  time_t;
#endif
#ifndef __STAT
typedef     short dev_t;
typedef     long  off_t;
#endif
typedef     u_short     uid_t;
```

```
typedef     u_short     gid_t;

#define     NBBY  8           /* number of bits in a byte */
/*
 * Select uses bit masks of file descriptors in longs.
 * These macros manipulate such bit fields (the filesystem macros use chars).
 * FD_SETSIZE may be defined by the user, but the default here
 * should be >= NOFILE (param.h).
 */
#ifndef     FD_SETSIZE
#define     FD_SETSIZE  256
#endif

typedef long      fd_mask;
#define NFDBITS   (sizeof(fd_mask) * NBBY)      /* bits per mask */
#ifndef howmany
#define     howmany(x, y)     (((x)+((y)-1))/(y))
#endif

typedef     unsigned short fd_channel;

typedef     struct fd_set {
      fd_mask           fds_bits[howmany(FD_SETSIZE, NFDBITS)];
      fd_channel  fds_chan[howmany(FD_SETSIZE, NFDBITS)][NFDBITS];
} fd_set;

#define     FD_SET(n, p)      _$fdset(n, p)
#define     FD_CLR(n, p)      _$fdclr(n, p)
#define     FD_ISSET(n, p)    (n == 0 ? n : _$fdisset(n, p))
#define FD_ZERO(p)      _$fdzero(p)

#endif
```

APPENDIX B

C FUNCTIONS

```
/*****************************************************************************
*/
/*  Program:  commo2.c  VAX 6410
*/
/*  Function:  Program to create a server/client.
*/
/*  Compilation:  $cc commo.c
/*                $define lnk$library sys$library:vaxcrtl.olb
/*                $link commo, dua0:[netdist.lib]twglib/lib
/*  Execution if server:  commo
*/
/*  Execution if client:  commo host
*/
/*****************************************************************************
*/
#include "commo.h"
#include <ssdef>
#include <stdio>
#include <descrip>
int av_msgsock;
int start_indicator;
extern struct
 test3 {
  char sndfdl[12];/*0:11*/
  char rcvfdl[12];/*0:11*/
  char col_sw[4]; /*0:3*/
  int sdrop_track; /*4*/
  int stn;    /*4*/
  int ready;      /*4*/
  int sndflag;    /*4*/
  int rcvflag;      /*4*/
  int retrieving; /*4*/
 }aanetcommo;
main()
{
/* Function declaration*/
      int assign_keyboard(void);
      int connect_to_host(int,char *,int);
      int wait_for_connection(int,int);
      int create_sock(void);
      int establish_connection(int,char *,int,int);
      struct list *retrieve(void);
      void check_usage(int,char *);
      void closedown(char[]);
      void insert(union test2 *);
      void check_environment(int,fd_set,struct timeval,union test2 *);
      void print_list(struct list *);
      void create_global(void);
/*.....................................................................*
/

      int argc;
      int i;
      int keyboard;
      int pms_sock, av_sock, ad_sock;
      int status;
```

```
      char *pms_argv[1],*av_argv[1],*ad_argv[1];
      $DESCRIPTOR (cluster,"SWFLG3");
      extern int uerrno;
      fd_set readchans;
      struct timeval timeout;

/*.....................................................................
*/
      status = SYS$ASCEFC(96,&cluster,0,0);
/*    if (status != SS$_NORMAL) LIB$STOP(status);*/

      create_global();
/* Setting up the environment for network */
/*    keyboard = assign_keyboard();*/

/*  Connect to AVTOC as a client */
      argc = 2;
      av_argv[1] = "hel-iris2";
      av_sock = create_sock();
      av_msgsock = establish_connection(av_sock,av_argv[1],argc,port3);

/* Initialize values */
      status = SYS$CLREF(120);
      start_indicator = 1;
      aanetcommo.sdrop_track = 0;
      aanetcommo.stn = 0;
      timeout.tv_sec = 2;
      timeout.tv_usec = 0;
      head_pointer = 0;
      tail_pointer = &head;

/* Clear the buffer */
            for (i=0; i<12; ++i) buffer00.buffer[i] = 0x0;
      while (1)
       {
        check_environment(keyboard, readchans, timeout, &buffer00);
       } /* while (1) */
}
/*****************************************************************************
*/
/*                    Function assign_keyboard
*/
/*****************************************************************************
*/
int assign_keyboard(void)
/* Open up an fd for the keyboard */
{
      int keyboard;
      if (keyboard = open("SYS$INPUT:",O_RDWR) <0)
       {
        perror("Error opening SYS$INPUT:");
        exit(1);
       }
      return(keyboard);
}
/*****************************************************************************
*/
/*                    Function to de-assign the socket
```

```
*/
/*****************************************************************************
*/
void closedown(char message[])
{
      int msgsock;
      printf("%s\n",message);
      netclose(msgsock);
      exit(1);
}
/*****************************************************************************
*/
/*                    Function to connect to remote host
*/
/*****************************************************************************
*/
int connect_to_host(int sock,char *hostname,int port)
{
      struct hostent *hp;
      struct sockaddr_in sin;

/* Get the host address of the remote server */
      hp = gethostbyname(hostname);
      if (hp == 'NUL')
       {
        printf ("%s: unknown host\n",hostname);
        exit(2);
       }

/*initialize socket address structure*/
      bzero((char *) &sin,sizeof (sin));
      bcopy((char *)hp->h_addr, (char *) &sin.sin_addr, hp->h_length);
      sin.sin_family = AF_INET;
      sin.sin_port = htons(port);

/* Connect to the socket at the remote host */
      if (connect(sock, &sin, sizeof(sin))<0)
       {
        netclose(sock);
        perror("netecho: connect");
        exit(5);
       }

/*    printf("Open\r\nConnection established to '%s'.\r\n",hostname);*/
      return(sock);
}
/*****************************************************************************
*/
/*                    Function to wait for a connection
*/
/*****************************************************************************
*/
int wait_for_connection(int sock,int port)
{
      int msgsock;
      int length;
      struct sockaddr_in sin;
```

```
/*initialize socket address stucture*/
      bzero((char*) &sin, sizeof(sin));
      sin.sin_family = AF_INET;
      sin.sin_port = htons(port);
      sin.sin_addr.s_addr = INADDR_ANY;

/*bind socket data structure to this socket*/
      if (bind (sock,&sin,sizeof(sin)))
       {
        perror("Error binding socket");
        netclose(sock);
        exit(1);
       }

/*prepare socket queue for connection requests and accept connections*/
      listen(sock,5);

      length = sizeof(sin);
      msgsock = accept(sock,&sin,&length);
      if (msgsock < 0)
       {
        perror("accept");
        netclose(sock);
        exit(1);
       }

      netclose(sock);
      return(msgsock);
}
/*****************************************************************************
*/
/*                    Function check_usage
*/
/*****************************************************************************
*/
void check_usage(int argc,char *command)
{
      if (argc>2)
       {
            printf ("Usage: %s hostname\n", command);
            exit(1);
       }
}
/*****************************************************************************
*/
/*                    Function create_sock
*/
/*****************************************************************************
*/
int create_sock(void)
/* Create "local" socket, which will be used for the connection */
{
      int sock;
      sock = socket(AF_INET, SOCK_STREAM, 0);
      if (sock < 0)
       {
            perror ("netecho: socket");
            exit(3);
```

```
        }
      return(sock);
}
/*****************************************************************************
*/
/*                    Function establish_connection
*/
/*****************************************************************************
*/
int establish_connection(int sock,char *hostname,int argc,int port)
/*connecting to remote host if argc equals 2 (client) or
  waiting for the connection if argc equals 1 (server)*/
{
      int msgsock;
      if (argc==2)
       {
        msgsock = connect_to_host(sock, hostname,port);
       }
      else
       {
/*       printf("Trying...\n");*/
         msgsock = wait_for_connection(sock,port);
       }
      return(msgsock);
}
/*****************************************************************************
*/
/*                    Function insert
*/
/*  Insert messages into the linked list.
*/
/*  Call format:  insert(&buffer00);
*/
/*****************************************************************************
*/
void insert(union test2 *buffer_arg)
{
      int length, i;
/*.....................................................................
*/
      if (head_pointer == 0)
       {
        head_pointer = &head;
        tail_pointer = &head;
        first_insert = 1;
       }
      if (first_insert == 1)
       {
        tail_pointer->next = (struct list *) calloc(1,sizeof (struct list));
        tail_pointer->next = 0;
        first_insert = 0;
       }
      else
       {
        tail_pointer->next = (struct list *) calloc(1,sizeof (struct list));
        tail_pointer = tail_pointer->next;
        tail_pointer->next = 0;
       }
```

```
        length = sizeof (buffer_arg->buffer);
        for (i=0; i<length; ++i)
         tail_pointer->data.buffer[i] = buffer_arg->buffer[i];
}
/*****************************************************************************
*/
/*                    Function retrieve
*/
/*  Function to retrieve messages from the linked list by application
program.*/
/*  Call format:  current_pointer = retrieve();
*/
/*****************************************************************************
*/
struct list *retrieve(void)
{
      struct list *current_pointer;

      current_pointer = head_pointer;
      head_pointer = head_pointer->next;
      current_pointer->next = 0;
      return(current_pointer);
}
/*****************************************************************************
*/
/*                    Function check_environment
*/
/*  Function to check if there is an incoming message or an outgoing message.
*/
/*  Incoming messages are detected through the network socket.
*/
/*  This function should be called in a loop throughout the simulation.
*/
/*****************************************************************************
*/
void check_environment(int keyboard,fd_set readchans,
                       struct timeval timeout,union test2 *buffer_arg)
/* Arguments definition */
/* keyboard - keyboard channel
   readchans - channels to be checked for activity
   timeout - How long should it wait each time it checks the channel
   start_indicator - 0 indicates start simulation, set by AVTOC
   buffer_arg - Pointer to the test2 buffer union*/
{
        int nfound, i, buflen, status;
        int action;
        int cnt1, cnt2, cnt3, done;
        char selection;
        char temp_buffer[12];
        int SYS$SETEF(), SYS$ASCEFC();
        int SYS$WAITFR(), SYS$CLREF();

/*      status = SYS$WAITFR(115);*/
/*      if (!(status&1)) LIB$STOP(status);*/
/*      status = SYS$CLREF(115);*/
/*      if (!(status&1)) LIB$STOP(status);*/

        buflen = sizeof(buffer00);
```

```
/*add sock to fd*/
        FD_SET(av_msgsock,&readchans);

/*look for i/o events*/
        nfound = 0;
        printf("av_msgsock = %x before nselect\n",av_msgsock);
        printf("nfound = %x before nselect\n",nfound);
        nfound = nselect(FD_SETSIZE, &readchans, 0, 0, &timeout);
        printf(" ");
        printf("av_msgsock = %x after nselect\n",av_msgsock);
        printf("nfound = %x after nselect\n",nfound);
        printf("start_indicator = %x\n",start_indicator);
        printf(" ");
        if (nfound == -1)
         {
          perror("select");
/*        exit(1);*/
          } /* (nfound == -1) */
/* end simulation when indicator set to 1*/
/*      if (start_indicator == 2)
         {
          status = SYS$CLREF(120);
          netclose(av_msgsock);
          exit(1);
         }*/
/* process all events */
        if (nfound)
         {
/* network AVTOC? */
          if (FD_ISSET(av_msgsock, &readchans))
           {
            FD_CLR(av_msgsock, &readchans);
            nfound--;
            cnt1=0;
            cnt2=0;
            done = 0;
            while(!done)
             {
              status = netread(av_msgsock,temp_buffer,buflen);
            cnt2=status+cnt2;
            cnt3=0;
            while(cnt1 <= cnt2-1)
             {
              buffer_arg->buffer[cnt1]=temp_buffer[cnt3];
              cnt3++;
              cnt1++;
             }
              if (cnt2 == buflen)
             {
              done=1;
             }
             }/* done */
           } /* av_msgsock */
/* Shifting bytes */
            for (i=0; i<12; ++i)
             temp_buffer[11-i] = buffer_arg->buffer[i];
            for (i=0; i<12; ++i)
             buffer_arg->buffer[i] = temp_buffer[i];
```

```
/* Check what is in the buffer */
            if (buffer_arg->buffer26.sublabel==0x18)
             {
              start_indicator = buffer_arg->buffer26.start_indicator;
            if (start_indicator == 1)
               {
/*            start_indicator = 2;*/
             }
            else  /* is 0 */
             {
              status = SYS$SETEF(120);
              if (!(status&1)) LIB$STOP(status);
               }
              }
              else if (start_indicator == 0)
               {
/* Store information received in an array for Neal to use*/
               insert(buffer_arg);
                 aanetcommo.rcvflag = 1;
               }  /*start_indicator */
/* Clear the buffer */
            for (i=0; i<12; ++i) buffer_arg->buffer[i] = 0x0;
          } /* (nfound >0) */

/* Any data to be sent? */
          if (aanetcommo.sndflag==1)
           {
            aanetcommo.sndflag = 0;
/* Moving bytes to pointer */
            for (i=0; i<12; ++i)
             buffer_arg->buffer[i] = aanetcommo.sndfdl[i];
/* Shifting bytes before sent */
            for (i=0; i<12; ++i)
             temp_buffer[11-i] = buffer_arg->buffer[i];
            for (i=0; i<12; ++i)
             buffer_arg->buffer[i] = temp_buffer[i];
/* Sending information over the net */
           status=netwrite(av_msgsock,buffer_arg->buffer,buflen);
           if(status != buflen) closedown("netecho: send failed");
/* Clear the buffer */
           for (i=0; i<12; ++i) buffer_arg->buffer[i] = 0x0;
           if (start_indicator == 1)
            {
             start_indicator = 2;
            }
           }
/* Checking if program test wants to retrieve a message */
          if (head_pointer == 0)
           {
            aanetcommo.rcvflag = 0;
           }
          else if (aanetcommo.retrieving == 1)
           {
            aanetcommo.retrieving = 0;
            current_pointer = retrieve();
            if (head_pointer == 0) aanetcommo.rcvflag = 0;
            for (i=0; i<12; ++i)
             {
```

```
            aanetcommo.rcvfdl[i] = current_pointer->data.buffer[i];
            }
            aanetcommo.ready = 1;
          }

/* Check to see if there is a dropped track */
         if (aanetcommo.sdrop_track == 1)
          {
           aanetcommo.sdrop_track = 0;
           buffer_arg->buffer4.sublabel = 4;
           buffer_arg->buffer4.action = 5;
           buffer_arg->buffer4.pritrack = aanetcommo.stn;
           buffer_arg->buffer4.sourceid = 0x81;
/* Shifting bytes before sent */
            for (i=0; i<12; ++i)
             temp_buffer[11-i] = buffer_arg->buffer[i];
            for (i=0; i<12; ++i)
             buffer_arg->buffer[i] = temp_buffer[i];
/* Sending information over the net */
           status=netwrite(av_msgsock,buffer_arg->buffer,buflen);
           if(status != buflen) closedown("netecho: send failed");
/* Clear the buffer */
           for (i=0; i<12; ++i) buffer_arg->buffer[i] = 0x0;
          }
}
/*****************************************************************************
/
/*                        Function to print the list of messages received
*/
/* Call print_list(head_pointer);                                            */
/*****************************************************************************
/
void print_list(struct list *current_pointer)
{
      int i=0;
      while (current_pointer !=0)
       {
        ++i;
        printf("%d) %x\n",i,current_pointer->data.buffer[0]);
        current_pointer = current_pointer->next;
       }
}
```

APPENDIX C

LINKER FILES

```
$!.....................commo2.com...................!
$set verify
$define lnk$library sys$library:vaxcrtl.olb
$link/map commo2, create_global, linker/opt,-
[hac.exe]getchan, dua0:[netdist.lib]twglib/lib
$set noverify
```

```
!.....................linker.opt...................!
sys$share:vaxcrtl.exe/share
PSECT=aanetcommo,PIC,OVR,REL,GBL,SHR,NOEXE,WRT,PAGE
```

APPENDIX D

FORTRAN SUBROUTINE

```
C*******************************************************************************
**
      Subroutine Create_global      !23-May-1991
C   Author:  Maria del C. Lopez                  Computer:  VAX 6410
C
C    This subroutine creates the global section to be used by the process
commologic, startflights, message and commo2.
c    program.
c
c   Directory:
c     TAURUS::[lopez]
c
c   External subroutines called:
c     None
c
c   Files opened:
c     netcommo.dat------------------'unknown'
c
c   Global section name:
c     netcommo
c
c   Common section name:
c     aanetcommo
c
c   Compile only since it is a subroutine:
c     $for create_global
c*******************************************************************************
**
      IMPLICIT NONE

      INCLUDE '($SECDEF)'
      INCLUDE '($SSDEF)'
      INCLUDE '($IODEF)'
C   Global section.
      INTEGER*4 GET_CHAN
      EXTERNAL GET_CHAN
      INTEGER*2   sec_chan
      COMMON /ACHANNEL/sec_chan

      CHARACTER*8       globsec /'NETCOMMO'/

      INTEGER*4 SYS$CRMPSC,STATUS
      INTEGER*4 SEC_FLAGS,MAPRANGE(2),RETADR(2)

c...Global variable
      byte  sndfdl(0:11)
      byte  rcvfdl(0:11)
      CHARACTER*4 col_sw
      INTEGER*4   sdrop_track
      INTEGER*4   stn
      INTEGER*4   ready
      INTEGER*4   rcvflag
      INTEGER*4   sndflag
      INTEGER*4   retrieving

      COMMON /aanetcommo/ sndfdl, rcvfdl, col_sw, sdrop_track,
```

```
     &                    stn,ready, sndflag,
     &                    rcvflag, retrieving
c.....................................................................
.c
C  Create section to share data with processes commologic, startflights and
message.

      sec_flags = SEC$M_GBL .OR. SEC$M_WRT .OR. SEC$M_DZRO

      OPEN(UNIT=1,FILE='NETCOMMO.DAT',USEROPEN=GET_CHAN,SHARED,
     &       INITIALSIZE=1,STATUS='UNKNOWN')

      maprange(1) = %LOC(sndfdl)
      maprange(2) = %LOC(retrieving)

      status = SYS$CRMPSC(maprange,retadr,,%VAL(sec_flags),
     &                    globsec,,,%VAL(sec_chan),%VAL(1),,,)
      IF (.NOT. status) CALL LIB$STOP(%VAL(status))

      RETURN
      END
```